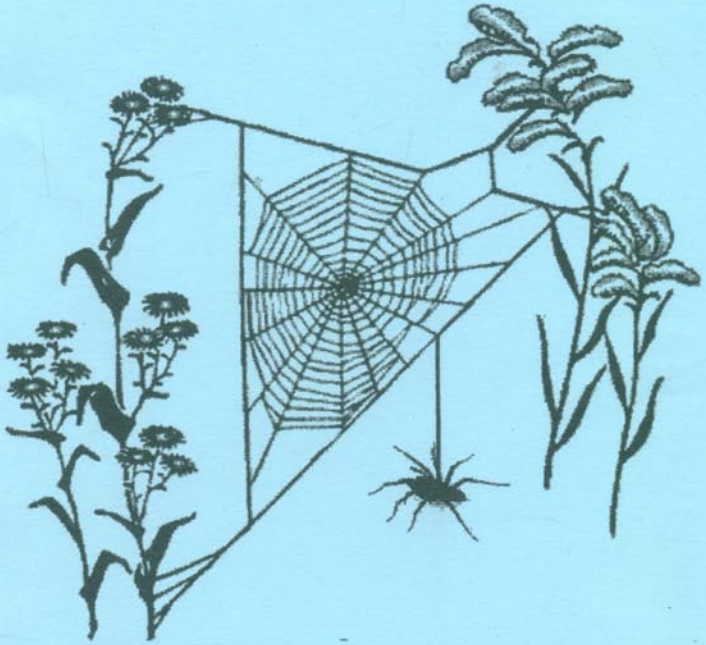
А. В. Дудатьєв, О. П. Войтович, В. А. Каплун

ЗАХИСТ КОМП'ЮТЕРНИХ МЕРЕЖ
ТЕОРІЯ ТА ПРАКТИКА



Міністерство освіти і науки України
Вінницький національний технічний університет

А. В. Дудатьєв, О. П. Войтович, В. А. Каплун

ЗАХИСТ КОМП'ЮТЕРНИХ МЕРЕЖ ТЕОРІЯ ТА ПРАКТИКА

Навчальний посібник

Вінниця
ВНТУ
2010

УДК 681.3.067
ББК 32.973.202
Д81

Рекомендовано до друку Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 3 від 30.10.2008)

Рецензенти:
О. Д. Азаров, доктор технічних наук, професор
О. М. Роїк, доктор технічних наук, професор
Б. П. Авксентюк, доктор технічних наук, професор

Дудатьєв, А. В.

Д81 Захист комп'ютерних мереж. Теорія та практика : навчальний посібник \ А. В. Дудатьєв, О. П. Войтович, В. А. Каплун. – Вінниця : ВНТУ, 2010. – 219 с.

В посібнику розглянуті основні поняття та засади забезпечення безпеки комп'ютерних мереж. Подані теоретичні відомості та матеріал для проведення практичних і лабораторних робіт з дисципліни "Захист комп'ютерних мереж". Посібник розрахований у першу чергу на студентів, які навчаються за напрямком "Інформаційна безпека" та всіх фахівців, які цікавляться питаннями захисту комп'ютерних мереж.

УДК 681.3.067
ББК 32.973.202

ЗМІСТ

ВСТУП

Новий підхід до питання безпеки виник в результаті появи розподілених систем обробки даних, в яких мережі та комунікаційне обладнання використовується для обміну даних між терміналами користувача та центральними комп'ютерами або між будь-якими комп'ютерами взагалі. В зв'язку з цим виникла необхідність захисту мереж, по яких передаються дані, а разом з нею з'явився термін мережева безпека. При цьому мається на увазі не одна окремо взята локальна мережа, а деяка сукупність мереж підприємств, службових організацій та навчальних закладів, зв'язаних між собою в об'єднану мережу обробки даних. Між двома формами безпеки – комп'ютерної та мережевої навряд чи можна провести чітку границю. Наприклад, одним із широковідомих типів втручання в роботу інформаційної безпеки є комп'ютерний вірус. Вірус може вноситись в систему як фізично, наприклад з зовнішніх носіїв, так і через об'єднану мережу.

Дана книга присвячена питанням захисту інформації і безпеки її передавання в об'єднаних мережах, методам виявлення та запобігання можливим порушенням захисту, а також ліквідації негативних наслідків таких порушень. Це загальне формулювання, охоплює сукупність найрізноманітніших аспектів. Начальний посібник складається із двох частин.

У першій частині розглянуті основні методи атак на комп'ютерні мережі та поданий теоретичний матеріал, який охоплює практично весь спектр направлень щодо захисту комп'ютерних мереж: на рівні *IP* протоколів, захист веб-додатків, захист систем мережевого керування, розглянутий захист поштових повідомлень, а також захист від шкідливого програмного забезпечення. Захист на рівні *IP* охоплює три сфери безпеки: автентифікацію, конфіденційність та управління ключами. Механізм автентифікації повинен гарантувати, що отримане повідомлення було насправді відправлене стороною, вказаною як джерело повідомлення та не було змінено. Засоби конфіденційності повинні забезпечувати можливість шифрування повідомлень, щоб виключити ризик їх читання третьою стороною. Захист на рівні веб-додатків включає в себе використання протоколів безпечного передавання даних по мережі, таких як *SSL/TLS*, *IPSec*. Для правильного захисту та діагностування мережі, можна скористатися стандартними засобами антивірусного захисту.

У другій частині запропоновані теми практичних занять, які містять задачі з відповідних тем та набір лабораторних робіт, виконання яких дозволить набути практичних навичок та закріпити теоретичні знання.

Автори висловлюють подяку колективу кафедри захисту інформації за надану допомогу в написанні навчального посібника і особисто завідувачу кафедри д.т.н., проф. Лужецькому В. А.

1 МЕТОДИ ЗЛАМУ КОМП'ЮТЕРНИХ МЕРЕЖ

1.1 Історія методів зламу

Цей розділ присвячено еволюції методів зламу комп'ютерних систем та мереж. Багатьох випадків успішного злому можна було уникнути при правильному налаштуванні системи і використанні програмних методів.

1.1.1 Колективний доступ

Початковою метою при створенні *Інтернету* був спільний доступ до даних і спільна робота дослідницьких інститутів. Таким чином, більшість систем були сконфігуровані для колективного використання інформації. При роботі в операційній системі *Unix* використовувалася мережева файлова система (*Network File System, NFS*), яка дозволяла одному комп'ютеру підключати диск іншого комп'ютера через локальну мережу (*local area network, LAN*) або *Інтернет*.

Цим механізмом скористалися перші зловмисники для діставання доступу до інформації – вони підключали віддалений диск і прочитували її. *NFS* використовувала номери ідентифікаторів користувача (*user ID, UID*) як проміжну ланку для доступу до даних на диску. Якщо користувач *A* з номером *ID* 104 мав дозвіл на доступ до файлу на своєму домашньому комп'ютері, то інший користувач *B* з номером *ID* 104 на видаленому комп'ютері також міг прочитати цей файл. Небезпека зросла, коли деякі системи вирішили загальний доступ в кореневу файлову систему (*root file system*), включаючи файли конфігурації і паролів. В цьому випадку зловмисник міг оволодіти правами адміністратора і підключити кореневу файлову систему, що дозволяло йому змінювати файли конфігурації віддаленої системи.

Загальний доступ до файлів не можна вважати вразливим місцем, це, скоріше, серйозна помилка конфігурації. Найцікавіше, що багато операційних систем (включаючи ОС *Sun*) поставляються з кореневою файловою системою, що експортується для загальнодоступного читання/запису. Отже, будь-який користувач на будь-якому комп'ютері, зв'язавшись із *Sun*-системою, може підключати кореневу файлову систему і вносити до неї довільні модифікації. Якщо не змінити задану за замовчуванням конфігурацію системи, то це може зробити кожен, хто захоче.

В більшості випадків сумісний доступ до файлів контролюється настройкою правил на зовнішньому міжмережевому екрані організації.

Вразливе місце у вигляді спільного доступу до файлів є не тільки в операційній системі *Unix*, але і в *Windows*. Деякі з цих систем можна налаштувати на дозвіл віддаленого підключення до їх файлових систем. Якщо користувач вирішить встановити спільний доступ до файлів, то він помилково може легко відкрити свою файлову систему для загального використання.

У ту ж саму категорію, що і спільне використання файлів і неправильна настройка, відноситься віддалений доступ з довірчими відносинами. Використання служби віддаленого входу в систему без пароля *rlogin* є звичайною справою серед системних адміністраторів і користувачів. *Rlogin* дозволяє користувачам входити відразу в декілька систем без повторного введення пароля. Цими дозволами управляють файли типу *.rhost* і *host*. У разі правильної настройки вони однозначно визначають ті системи, для яких дозволений віддалений вхід. Система *Unix* використовує знак "плюс" ("+"), розміщений в кінці файлу, який означає, що окремі системи довіряють користувачеві, що йому не обов'язково повторно вводити пароль, і не важливо, з якої системи він входить. Природно, зловмисники люблять знаходити такі помилки. Все, що їм потрібно зробити для проникнення в систему – це ідентифікувати обліковий запис користувача або адміністратора.

Міжмережеві екрани можуть контролювати не тільки колективне використання файлів, але і віддалений довірчий доступ із зовнішньої мережі. Проте у внутрішній мережі зовнішній міжмережевий екран такий контроль виконати не зможе. І це є серйозною проблемою безпеки

1.1.2 Слабкі паролі

Напевно, найзагальніший спосіб, який використовують зловмисники для входу в систему, – це слабкі паролі. Паролі як і раніше застосовуються для автентифікації користувачів. Оскільки це стандартний метод автентифікації для більшості систем, він не пов'язаний з додатковими витратами. Крім того, користувачі розуміють, як працювати з паролями. На жаль, багато хто не знає, як вибрати сильний пароль. Дуже часто використовуються короткі паролі (менше чотирьох символів) або легко вгадувані. Короткий пароль дозволяє застосувати атаку "в лоб", тобто зловмисник перебиратиме паролі, поки не підбере потрібний.

Пароль, який легко вгадати, також є слабким паролем. Наприклад, пароль кореневого каталогу "*toor*" ("*root*", записаний в зворотному порядку) дозволить зловмисникові дуже швидко дістати доступ до системи. Деякі проблеми, пов'язані з паролем, належать до категорії неправильного налаштування системи. Наприклад, в старій корпорації цифрового обладнання *Digital Equipment Corporation* систем *VAX/VMS* обліковий запис служби експлуатації (*field service*) мав ім'я "*field*" і заданий за замовчуванням пароль "*field*". Якщо системний адміністратор не знав про це і не змінював пароль, то будь-хто міг дістати доступ до системи за допомогою даного облікового запису.

Наочний приклад того, як слабкі паролі допомагають зламувати системи – черв'як Моріса. У 1988 р. студент Роберт Моріс розробив програму, яка розповсюджувалася через *Інтернет*. Ця програма використовувала декілька вразливих місць для отримання доступу до комп'ютерних

систем і відтворення самої себе. Одним з вразливих місць були слабкі паролі. Програма разом із списком найбільш поширених паролів використовувала такі паролі: порожній пароль, ім'я облікового запису, додане до самого себе, ім'я користувача, прізвище користувача і зарезервоване ім'я облікового запису. Цей черв'як завдав збитків достатньо великій кількості систем і вельми ефективно вивів з ладу *Інтернет*.

Альтернативою паролям є смарт-карти (маркери автентифікації) і біометрія. Проте розгортання таких систем пов'язане з додатковими витратами. Крім того, їх можна використовувати не завжди. Наприклад, онлайновий продавець навряд чи скористається ними для автентифікації своїх покупців. Отже, швидше за все, паролі ще довго будуть використовуватись.

Не існує універсальних вирішень проблеми паролів. У більшості операційних систем системний адміністратор має можливість настроювати вимоги до паролів, і це дуже важливо. Проте кращий захист від слабких паролів – навчання службовців належному розумінню проблем безпеки.

1.1.3 Дефекти програмування

Зловмисники користувалися дефектами програмування багато раз. До цих дефектів відноситься залишений в програмі "Чорний хід" (*back door*), який дозволяє згодом входити в систему. Ранні версії програми *Sendmail* мали такі "чорні ходи". Найчастіше використовувалася команда *WIZ*, доступна в перших версіях *Sendmail*, що працюють під *Unix*. При підключенні за допомогою *Sendmail* (через мережевий доступ до порту 25) і введення команди *WIZ* з'являлася можливість запуску інтерпретатора команд (*root shell*). Ця функція спочатку була включена в *Sendmail* як інструмент для налагодження програми. Подібні функції, залишені в програмах загального призначення, дозволяють зловмисникам вмить проникати в системи, що використовують ці програми. Зловмисники ідентифікували безліч таких лазівок, більшість з яких була, у свою чергу, усунено програмістами. На жаль, деякі "чорні ходи" існують до цих пір, оскільки не на всіх системах оновилося програмне забезпечення.

Не так давно бум в програмуванні веб-сайтів привів до створення нової категорії "необережного" програмування. Вона має відношення до онлайнової торгівлі. На деяких веб-сайтах інформація про покупки: номер товару, кількість і навіть ціна – зберігається безпосередньо в рядку адреси *URL*. Ця інформація використовується веб-сайтом, при підрахуванні вартості покупок і визначенні, скільки грошей знято з кредитної карти. Виявляється, що багато сайтів не перевіряють інформацію при впорядковуванні списку, а просто беруть її з рядка *URL*. Якщо зловмисник модифікує *URL* перед підтвердженням, він зможе отримати порожній номер в списку. Бували випадки, коли зловмисник встановлював ціну з від'ємним значенням і, замість того щоб витратити гроші на покупку, одержував від веб-сайта

кредит. Не дивлячись на те, що це вразливе місце не дозволяє зловмисникові увійти до системи, великому ризику піддається і веб-сайт, і організація.

1.1.4 Соціальний інжиніринг

Соціальний інжиніринг – це отримання несанкціонованого доступу до інформації або до системи без застосування технічних засобів. Замість використання вразливих місць зловмисник грає на людських слабостях. Найсильніша зброя зловмисника в цьому випадку – приємний голос і акторські здібності. Зловмисник може подзвонити по телефону співробітникові компанії під виглядом служби технічної підтримки і дізнатися його пароль "для вирішення невеликої проблеми в комп'ютерній системі".

Іноді зловмисник під виглядом службовця компанії дзвонить в службу технічної підтримки. Якщо йому відоме ім'я службовця, то він говорить, що забув свій пароль, і в результаті або дізнається пароль, або міняє його на потрібний. Враховуючи, що служба технічної підтримки орієнтована на невідкладне надання допомоги, вірогідність отримання зловмисником хоч би одного облікового запису вельми велика.

Зловмисник не полінується зробити безліч дзвінків, щоб як слід вивчити свою мету. Він почне з того, що дізнається імена керівників на веб-сайті компанії. За допомогою цих даних він спробує роздобути імена інших службовців. Ці нові імена стануть в нагоді йому в розмові із службою технічної підтримки для отримання інформації про облікові записи і про процедуру надання доступу. Ще один телефонний дзвінок допоможе дізнатися, яка система використовується і як здійснюється видалений вхід в систему. Використовуючи імена реальних службовців і керівників, зловмисник придумає цілу історію про важливу нараду на сайті клієнта, на яку він нібито не може потрапити зі своїм обліковим записом віддаленого доступу. Співробітник служби технічної підтримки зіставить факти: людина знає, що відбувається, знає ім'я керівника і компанії – і, недовго думаючи, надасть йому доступ.

Іншими формами соціального інжинірингу є дослідження сміття організацій, віртуальних сміттєвих корзин, використання джерел відкритої інформації, відкритий грабіж і самозванство, крадіжка портативного комп'ютера або набору інструментів. Інструменти допоможуть йому зіграти роль обслуговуючого персоналу або співробітника компанії.

Соціальний інжиніринг дозволяє здійснити найхитромудріші проникнення, але вимагає часу і таланту. Він зазвичай використовується зловмисниками, які намітили своєю жертвою конкретну організацію.

Найкращою обороною проти атак соціального інжинірингу є інформування службовців. Необхідно розробити правила, за якими служба технічної підтримки може вступати в контакт з користувачами.

1.1.5 Переповнювання буфера

Переповнювання буфера – це одна з помилок програмування, яка використовується зловмисниками. Переповнювання буфера важче виявити, ніж слабкі паролі або помилки конфігурації. Проте потрібно зовсім трохи досвіду для його експлуатації. На жаль, зловмисники, які відшукали можливості переповнювання буфера, публікують свої результати, включаючи в них сценарій експлойту або програму, яку може запустити кожен, хто має комп'ютер.

Переповнювання буфера особливе небезпечне тим, що дозволяє зловмисникам виконати практично будь-яку команду в системі. Більшість сценаріїв переповнювання буфера дають зловмисникам можливість створення нових способів проникнення в систему, що атакується. Слід зазначити, що переповнювання буфера не обмежує доступ до віддаленої системи. Існує декілька типів переповнювань буфера, за допомогою яких можна підвищити рівень користувача в системі. Локальні вразливі місця так само небезпечні (якщо не більше), як і віддалені. Переповнювання буфера – це спроба розмістити дуже багато даних в області комп'ютерної пам'яті. Наприклад, якщо ми створимо змінну завдовжки у вісім байтів і запишемо в неї дев'ять байтів, то дев'ятий байт розміститься в пам'яті відразу услід за восьмим. Якщо ми спробуємо помістити ще більше даних в цю змінну, то зрештою заповниться вся пам'ять, використовувана операційною системою.

Стек управляє перемиканням між програмами і повідомляє операційну систему, який код виконувати, коли одна частина програми (або функції) завершує своє завдання. У стеку зберігаються локальні змінні функції. При атаці на переповнювання буфера зловмисник поміщає інструкції в локальну змінну, яка зберігається в стеку. Ці дані займають в локальній змінній більше місця, чим виділений під неї об'єм, і переписують адресу в точку цієї нової інструкції. Ця нова інструкція завантажує для виконання програмну оболонку (що здійснює інтерактивний доступ) або інше застосування, змінює файл конфігурації (*inetd.conf*) і дозволяє зловмисникові доступ за допомогою створення нової конфігурації.

Переповнювання буфера відбувається дуже часто через помилки в додатку, коли призначені для користувача дані копіюються в одну і ту ж змінну без перевірки об'єму цих даних перед виконанням операції. Дуже багато програм страждають цим. Проте дана проблема усувається досить швидко, відразу, як тільки виявляється і привертає до себе увагу розробника. Але якщо це так легко зробити, то чому переповнювання буфера існує до цих пір? Якщо програміст перевірятиме розмір призначених для користувача даних перед їх розміщенням в заздалегідь оголошеній змінній, то переповнювання буфера можна буде уникнути.

Переповнювання буфера можна виявити в результаті дослідження початкового коду програм. Хоча це звучить просто, насправді процес

довгий і складний. Набагато легше пам'ятати про переповнювання буфера в процесі створення програми, ніж потім повертатися і шукати його.

Існує декілька автоматизованих сценаріїв, використовуваних для пошуку вірогідного переповнювання буфера. До таких програмних засобів відноситься програма *SPLINT* (http://lclint.cs.virginia.edu/), яка проводить перевірку коду перед його компіляцією.

1.1.6 Відмова в обслуговуванні

Атаки на відмову в обслуговуванні (*Denial-of-service, DoS*) – це зловмисні дії, що виконуються для заборони легальному користувачеві доступу до системи, мережі, додатку або інформації. Атаки *DoS* мають багато форм, вони бувають централізованими (запущеними від однієї системи) або розподіленими (запущеними від декількох систем).

Атакам *DoS* не можна повністю запобігти, їх не можна і зупинити, якщо не вдасться виявити джерело нападу. Атаки *DoS* відбуваються не тільки в віртуальному просторі. Пара кусачків є нехитрим інструментом для *DoS*-атаки – треба тільки узяти їх і перерізати кабель локальної мережі.

Слід зазначити ще один важливий момент в підготовці більшості атак. Поки зломщикові не вдасться проникнути в цільову систему, *DoS*-атаки запускаються з підроблених адрес. *IP*-протокол має помилку в схемі адресації: він не перевіряє адресу відправника при створенні пакета. Таким чином, зловмисник дістає можливість змінити адресу відправника пакета для приховання свого розташування.

1.1.7 Централізовані *DoS*-атаки

Першими типами *DoS*-атак були централізовані атаки (*single-source*), тобто для здійснення атаки використовувалася одна система. Найбільш широку популярність здобула так звана синхронна атака (*SYN flood attack*). При її виконанні система-відправник посилає величезну кількість *TCP SYN*-пакетів (пакетів з синхронізуючими символами) до системи-одержувача. *SYN*-пакети використовуються для відкриття нових *TCP*-з'єднань. При отриманні *SYN*-пакета система-одержувач відповідає *ACK*-пакетом, що повідомляє про успішний прийом даних, і посилає дані для установлення з'єднання до відправника *SYN*-пакета. При цьому система-одержувач поміщає інформацію про нове з'єднання в буфер черги з'єднань. У реальному *TCP*-з'єднанні відправник після отримання *SYN ACK*-пакета повинен відправити завершальний *ACK*-пакет. Проте в цій атаці відправник ігнорує *SYN ACK*-пакет і продовжує відправку *SYN*-пакетів. Зрештою буфер черги з'єднань на системі-одержувачі переповнюється, і система перестає відповідати на нові запити на підключення.

Очевидно, що якщо джерелом синхронної атаки виступає легальна *IP*-адреса, то його можна відносно легко ідентифікувати і зупинити атаку. А якщо адреса відправника є такою, що не маршрутизується, такою як

192.168.x.x? Тоді задача ускладнюється. У разі продуманого виконання синхронної атаки, і за відсутності належного захисту *IP*-адрес, зловмисника практично неможливо визначити.

Для захисту систем від синхронних атак було запропоновано декілька рішень. Найпростіший спосіб – розміщення таймера у всіх з'єднаннях, які чекають черги. Після закінчення деякого часу з'єднання повинні закриватися. Проте для запобігання грамотно підготовленій атаці таймер доведеться встановити рівним такому маленькому значенню, що це зробить роботу з системою практично неможливою. За допомогою деяких мережевих пристроїв можна виявляти і блокувати синхронні атаки, але ці системи схильні до помилкових результатів, оскільки шукають певну кількість відкладених підключень в заданому проміжку часу. Якщо атака має декілька джерел одночасно, то її дуже важко ідентифікувати.

Після синхронної атаки були виявлені і інші атаки, серйозніші, але менш складніші в запобіганні. При виконанні атаки "*ping* смерті" (*Ping of Death*) в цільову систему відправлявся *ping*-пакет (*ICMP* ехо-запит). У звичайному варіанті *ping*-пакет не містить даних. Пакет "*ping* смерті" містив велику кількість даних. При читанні цих даних системою-одержувачем відбувалося переповнювання буфера в стеку протоколів, і виникала повна відмова системи. Розробники стека не передбачали, що *ping*-пакет використовуватиметься так само, і тому перевірка кількості даних, що поміщаються в маленький буфер, не виконувалася. Проблема була швидко виправлена після виявлення, і в даний час залишилися мало систем, вразливих для цієї атаки.

"*Ping* смерті" – лише один різновид *DoS*-атаки, що націлений на вразливі місця систем або додатків і є причиною їх зупинення. *DoS*-атаки руйнівні лише в початковій стадії і швидко втрачають свою силу після виправлення системних помилок.

На жаль, виявлення нових *DoS*-атак, направлених проти додатків і операційних систем, носить регулярний характер. Від нових нападів можна трохи відпочити, лише поки зловмисники виправляють помилки в сценаріях минулих атак.

1.1.8 Розподілені *DoS*-атаки

Розподілені *DoS*-атаки (*Distributed DOS attacks*, *DDoS*) – це *DoS*-атаки, в здійсненні яких бере участь велика кількість систем. Зазвичай *DDoS*-атакою управляє одна головна система і один зловмисник. Ці атаки не обов'язково бувають складними. Наприклад, зловмисник відправляє *ping*-пакет по широкомовних адресах великої мережі, тоді як за допомогою підмінення адреси відправника – спуфінга (*spoofing*) – всі відповіді адресуються до системи-жертви. Така атака отримала назву *smurf*-атаки. Якщо мережа містить багато комп'ютерів, то кількість пакетів у відповідь, направлених до цільової системи, буде такою великою, що призведе до

виходу з ладу з'єднання через величезний об'єм даних, що передаються.

Сучасні *DDoS*-атаки стали витонченішими в порівнянні з *smurf*-атакою. Нові інструментальні засоби атак, такі як *Trinoo*, *Tribal Flood Network*, *Mstream* і *Stacheldraht*, дозволяють зловмисникові координувати зусилля багатьох систем в *DDoS*-атаці, направленій проти однієї мети. Ці засоби мають триланкову структуру. Зловмисник взаємодіє з головною системою або серверним процесом, розміщеним на системі-жертві. Головна система взаємодіє з підлеглими системами або клієнтськими процесами, встановленими на інших захоплених системах. Підлеглі системи ("зомбі") реально здійснюють атаку проти цільової системи. Команди, які передаються до головної системи і від головної системи до підлеглих, можуть шифруватися або передаватися за допомогою протоколів *UDP* (призначений для користувача протокол даних) або *ICMP* (протокол повідомлень, що управляють), залежно від використовуваного інструменту. Механізмом атаки, що діє, є переповнювання *UDP*-пакетами, пакетами *TCP SYN* або трафіком *ICMP*. Деякі інструментальні засоби випадковим чином змінюють адреси відправника атакуючих пакетів, надзвичайно утрудняючи їх виявлення.

Головним результатом *DDoS*-атак, що виконуються з використанням спеціальних інструментів, є координація великої кількості систем в атаці, спрямованій проти однієї системи. Незалежно від того, скільки систем підключено до *Інтернету*, скільки систем використовується для регулювання трафіка, такі атаки можуть буквально зупинити організацію, якщо в них бере участь достатня кількість підлеглих систем.

1.2 Вивчення сучасних методів

Багато сучасних атак виконуються так званими *script kiddies*. Це користувачі, що відшукують сценарії експлойтів в *Інтернет* та запускають їх проти всіх систем, які тільки можна знайти. Ці нехитрі способи атак не вимагають спеціальних знань або інструкцій.

Проте існують і інші методи, засновані на глибшому розумінні роботи комп'ютерів, мереж і систем, що атакуються. У даному розділі ми познайомимося з такими методами – з прослуховуванням (*сніфінгом*, від англ. *sniffing*) комутованих мереж та імітацією *IP*-адреси (*IP-spoofing*).

1.2.1 Прослуховування комутованих мереж

Прослуховування, або сніфінг (*sniffing*), використовується зловмисниками після злому системи для збирання паролів і іншої системної інформації. Для цього *сніфер* встановлює плату мережевого інтерфейсу в режим прослуховування змішаного трафіка (*promiscuous mode*), тобто мережевий адаптер перехоплюватиме всі пакети, що переміщаються по мережі, а не тільки пакети, адресовані даному адаптеру або системі. Сніфери такого типу добре працюють в мережах з

пропускною спроможністю, що розділяється, з мережевими концентраторами – хабами.

Сніфером (від англійського *sniff* – винюхувати) називають утиліти для перехоплення мережевого трафіка адресованого іншому вузлу або в більше загальному випадку – усього доступного трафіка, який проходить або не проходить через даний хост. Більшість сніферів являють собою цілком легальні засоби моніторингу та не потребують установлення додаткового устаткування. Проте, їхнє використання в загальному випадку незаконне й вимагає відповідних повноважень. До речі, слово *"sniffer"* є словом торговельної компанії *Network Associates*, що поширює мережевий аналізатор *"Sniffer Network Analyzer"*.

Об'єктом атаки може виступати: локальна мережа (як хабової, так і свічової архітектури), глобальна мережа, супутниковий і мобільний *Інтернет*, бездротові засоби зв'язку тощо.

За методом впливу на об'єкт існуючі атаки можна розділить на два типи: пасивні й активні. Пасивний сніфінг дозволяє перехоплювати лише ту частину трафіка, що фізично проходить через даний вузол. Все інше може бути отримано лише шляхом прямого втручання в мережеві процеси (модифікація таблиць маршрутизації, відправлення підроблених пакетів тощо).

Оскільки зараз більше використовуються мережеві комутатори, ефективність сніфінг почала знижуватися. У комутованому середовищі не застосовується режим широкомовної передачі, замість цього пакети відправляються безпосередньо до системи-одержувача. Проте комутатори не є захисними пристроями. Це звичайні мережеві пристрої, отже, забезпечувана ними безпека швидше побічний продукт їх мережевого призначення, ніж елемент конструкції. Тому цілком можлива поява сніфера, здатного працювати і в комутованому середовищі. Сніфер, спеціально розроблений для комутованого середовища, можна знайти за адресою http://ettercap.sourceforge.net/.

Для прослуховування трафіка в комутованому середовищі зловмисник повинен виконати одну з умов:

- "переконати" комутатор в тому, що трафік, який викликає інтерес, повинен бути спрямований до сніферу;
- змусити комутатор відправляти весь трафік до всіх портів.

При виконанні однієї з умов сніфер зможе прочитувати трафік і, таким чином, забезпечувати зловмисника шуканою інформацією

1.2.2 Перенаправлення трафіка

Комутатор спрямовує трафік до портів на підставі адреси доступу до середовища передачі даних (*Media Access Control – MAC*-адреса) – для кадру, який передається по мережі *Ethernet*. Кожна плата мережевого інтерфейсу має унікальну *MAC*-адресу, і комутатор "знає" про те, які

адреси призначені якому порту. Отже, при передачі кадру з певною *MAC*-адресою одержувача комутатор направляє цей кадр до порту, до якого приписана дана *MAC*-адреса.

Нижче наведені методи, за допомогою яких можна змусити комутатор направляти мережевий трафік до сніферу:

- *ARP*-спуфінг;
- дублювання *MAC*-адрес;
- імітація доменного імені.

ARP-спуфінг (*ARP-spoofing*). *ARP* – це протокол перетворення адрес (*Address Resolution Protocol*), використовуваний для отримання *MAC*-адреси, пов'язаної з певною *IP*-адресою. При передачі трафіка система-відправник посилає *ARP*-запит за *IP*-адресою одержувача. Система-одержувач відповідає на цей запит передачею своєї *MAC*-адреси, яка використовуватиметься системою-відправником для прямої передачі трафіка.

Якщо сніфер захопить трафік, що викликає у нього інтерес, то він відповість на *ARP*-запит замість реальної системи-одержувача і надасть власну *MAC*-адресу. В результаті система-відправник посилатиме трафік на сніфер.

Для забезпечення ефективності даного процесу необхідно переадресовувати весь трафік на сніфер замість реального місця призначення. Якщо цього не зробити, то з'явиться ймовірність виникнення відмови в доступі до мережі.

ARP-спуфінг працює тільки в локальних підмережах, оскільки *ARP*-повідомлення передаються тільки всередині локальної підмережі. Сніфер повинен розміщуватися в тому ж самому сегменті локальної мережі, що і системи відправника і одержувача.

Дублювання *MAC*-адреси системи-одержувача є ще одним способом "переконати" комутатор посилати трафік на сніфер. Для цього зловмисникові потрібно змінити *MAC*-адресу на сніфері і розміститися в системі, яка знаходиться в тому ж сегменті локальної мережі.

Вважається, що змінити *MAC*-адресу неможливо. Проте це зовсім не так. Це можна зробити в системі *Unix* за допомогою команди *ipconfig*. Аналогічні утиліти є і в системі *Windows*.

Для виконання *ARP*-спуфінга сніфер повинен розташовуватися в тій же самій локальній підмережі, що і обидві системи (відправник і одержувач), щоб мати можливість дублювання *MAC*-адреси.

Існує третій спосіб змусити комутатор відправляти весь трафік на сніфер: потрібно "обдурити" систему-відправника, щоб вона використовувала для передачі даних реальну *MAC*-адресу сніфера. Це здійснюється за допомогою імітації доменного імені.

При виконанні цієї атаки сніфер перехоплює *DNS*-запити від системи-відправника і відповідає на них. Замість *IP*-адрес систем, до яких був надісланий запит, система-відправник отримує *IP*-адресу сніфера і

відправляє весь трафік до нього. Далі сніфер повинен перенаправити цей трафік реальному одержувачеві.

Для забезпечення успіху даної атаки сніферу необхідно проглядати всі *DNS*-запити і відповідати на них до того, як це зробить реальний одержувач. Тому сніфер повинен розташовуватися на маршруті проходження трафіка від системи-відправника до *DNS*-сервера, а ще краще – в тій же локальній підмережі, що і відправник.

Сніфер міг би проглядати запити, що відправляються через *Інтернет*, але чим далі він від системи-відправника, тим складніше гарантувати, що він першим відповість на них.

1.2.3 Відправка всього трафіка до всіх портів

Замість виконання одного з вищеперерахованих методів зловмисник може змусити комутатор працювати як хаб (концентратор). Кожен комутатор використовує певний об'єм пам'яті для зберігання таблиці відповідностей між *MAC*-адресою і фізичним портом комутатора. Ця пам'ять має обмежений об'єм. У разі її переповнювання деякі комутатори можуть помилково видавати стан "відкритий". Це означає, що комутатор припинить передачу трафіка за певними *MAC*-адресами і почне пересилати весь трафік до всіх портів. В результаті комутатор почне працювати подібно до мережевого пристрою колективного доступу (хабу), що дозволить сніферу виконати свої функції. Для ініціалізації такого способу атаки зловмисник повинен безпосередньо підключитися до потрібного комутатора.

1.3 Викопання атак

У разі *ARP*-спуфінгу, дублювання *MAC*-адреси або *MAC*-флудингу зловмисник повинен безпосередньо підключитися до комутатора, що атакується. Таке підключення потрібне і для імітації доменного імені.

Висновок такий – зловмисник повинен встановити систему на локальному комутаторі. Він може спочатку увійти до системи через відому вразливість, а потім встановити необхідне для сніфінга програмне забезпечення. У іншому варіанті зловмисник вже знаходиться всередині організації (він її службовець або підрядчик). В цьому випадку він використовує свій законний доступ в локальну мережу, що дозволяє йому зв'язатися з комутатором.

Правильність *IP*-адреси в пакетах, які передаються по мережі, не перевіряється. Отже, зловмисник може змінити адресу відправника так, щоб здавалося ніби пакет прибуває з будь-якої адреси. Складність полягає в тому, що пакети (*SYN ACK*-пакети в *TCP*-з'єднанні) не зможуть повернутися до системи-відправника. Отже, спроба імітації *IP*-адреси (*IP-спуфінг*) для установлення *TCP*-з'єднання пов'язана з серйозними труднощами. Крім того, в *TCP*-заголовку міститься порядковий номер,

використовуваний для підтвердження прийому пакета. Початковий порядковий номер (*initial sequence number, ISN*) для кожного нового з'єднання вибирається псевдовипадковим чином.

У 1989 р. Стів Беловін (*Steve Bellovin*) з лабораторії *AT&T Bell* опублікував статтю "Проблеми безпеки сімейства протоколів *TCP/IP*" в журналі "Комп'ютери і мережі" ("*Computer and Communications Review*"). У цій статті йде мова про те, що в багатьох реалізаціях протоколів *TCP/IP* початковий порядковий номер не вибирається випадковим чином, а замість цього просто збільшується з певним приростом. Отже, за наявності даних про останній відомий номер *ISN* (початковий порядковий номер) можна обчислити наступний заздалегідь. Саме завдяки цьому можливе виконання атаки *IP*-імітації.

Спочатку зловмисник ідентифікує свою мету. Він повинен визначити величину приросту *ISN*. Це можна зробити, виконуючи серію легальних підключень до цільової системи і відзначаючи повернені номери *ISN* (при цьому зловмисник ризикує "засвітити" свою реальну *IP*-адресу).

Після визначення величини приросту *ISN* зловмисник посилає до цільової системи *TCP SYN*-пакет із зміненою *IP*-адресою відправника. Система відповість *TCP SYN ACK*-пакетом, який буде переданий за цією підробленою адресою і до зловмисника, отже, не дійде. *SYN ACK*-пакет містить початковий порядковий номер цільової системи. Для завершення процесу установлення підключення даний *ISN* необхідно підтвердити відправкою завершального *ACK*-пакета *TCP*. Зловмисник підраховує приблизний *ISN* і відправляє *ACK*-пакет, що містить підроблену *IP*-адресу відправника і підтвердження *ISN*.

Якщо все це буде правильно виконано, зловмисник закриє легальне підключення до цільової системи. Він зможе посилати команди і інформацію до системи, але не отримуватиме відповіді.

При атаці імітації *IP*-адреси комп'ютерна система вважає, що вона взаємодіє з іншою системою. Подібна атака, зазвичай, націлена на службу електронної пошти або веб-сервер, багато не дасть. Те ж саме справедливо і для атак "грубої сили" за допомогою командного рядка *telnet*. При настройці служб *rlogin* або *rsh* *IP*-адреса відправника є важливим компонентом, оскільки він визначає, кому дозволено використання цих служб. Віддалені хости, допущені до таких з'єднань, називаються довіреними. При використанні імітації *IP*-адреси довіреної системи можна успішно зламати цільову систему.

При виявленні системи, що має довірчі відносини з іншою системою і знаходиться в мережі, до якої можна підключитися, імітація *IP*-адреси дозволить зловмисникові дістати доступ в цю систему. Проте є ще одна проблема, яку він повинен вирішити. Цільова система у відповідь на підроблені пакети відправлятиме дані довіреній системі. Відповідно до специфікації *TCP* довірена система може відповісти перезавантаженням

або пакетом скидання (*RST*-пакетом), оскільки вона не має відомостей про підключення. Зловмисник не повинен допустити цього і зазвичай виконує *DoS*-атаку проти довіреної системи.

При підключенні до служби *rlogin* зловмисник може реєструватися як користувач довіреної системи (непоганим варіантом буде обліковий запис *root*, наявний у всіх *Unix*-системах). Пізніше він забезпечить собі зручніший спосіб входу в систему. (При підключенні за допомогою імітації *IP*-адреси зловмисник не отримує відповіді цільової системи на свої дії.) Він може налаштувати цільову систему для дозволу доступу за допомогою *rlogin* з віддаленої системи або додати обліковий запис для свого особистого використання.

1.4 Виявлення методів ненаправлених атак

Зловмисники, що використовують методи ненаправлених атак, не шукають певну інформацію або організацію: їм для злому підходить будь-яка система. Їх рівень кваліфікації коливається від дуже низького до найвищого, а як мотив виступає, перш за все, бажання привернути до себе увагу зломом будь-якої системи. Ймовірно, присутнє і жадання наживи, але що вони намагаються придбати таким чином – залишається загадкою.

Зловмисники, що використовують методи ненаправлених атак, відшукують будь-яку систему, яку можна знайти. Зазвичай у них немає певної мети. Іноді для пошуку вибирається мережа або доменне ім'я, але цей вибір, як правило, випадковий. Об'єктом атаки може стати будь-яка організація, до мережі якої зміг підключитися зловмисник.

Зловмисник по-різному проводить попереднє дослідження. Деякі починають атаку відразу ж, без всякої "розвідки" і точного визначення мети, якщо знаходять систему, підключену до мережі. Після проведення попереднього зондування атака зазвичай виконується із зламаних систем, щоб зловмисник міг "замести сліди".

Найчастіше зловмисники виконують приховане сканування діапазонну адрес, так зване половинне *IP*-сканування. З його допомогою виявляються системи, що знаходяться в даному діапазоні, і служби, доступні в цих системах. При прихованому скануванні виконується також розгорнена відправка *ping*-запитів в цьому діапазоні адрес, тобто відправка *ping*-запиту за кожною адресою і перегляд отриманих відповідей.

При виконанні прихованого сканування зловмисник зазвичай відправляє *TCP SYN*-пакет на *IP*-адресу і чекає на *TCP SYN ACK*-відповідь. При отриманні відповіді він посилає *TCP RST*-пакет для скидання з'єднання перш, ніж воно закриється. У багатьох випадках це дозволяє приховати спроби проникнення від служби реєстрації подій цільової системи.

Різновидом прихованого сканування є сканування із скиданням з'єднання (*reset scan*), при якому зловмисник посилає *TCP RST*-пакет на *IP*-

адресу. Зазвичай цей пакет не викликає ніяких дій у системі-одержувачі, і на нього не приходить відповідь. Проте якщо вказана система не існує, то маршрутизатор мережі, якій належить адреса одержувача, відповість *ICMP*-повідомленням: "*Хост недоступний*". Є інші способи сканування, що дають схожий результат. Слід відмітити, що сканування зі скиданням з'єднання виявляє системи, які знаходяться в мережі, але не дозволяє визначити виконувані на них служби, як це робить приховане сканування.

Існують способи прихованого сканування, що дозволяють визначити відкриті порти. Зазвичай вони виконуються за допомогою передачі трафіка до певних портів. Якщо порт закритий, він відповість *RST*-пакетом, інакше відповіді отримано не буде.

Іноді зловмисник виконує попереднє дослідження у декілька етапів. Спочатку він вибирає ім'я домену (зазвичай довільно) і починає зонну передачу *DNS*, направлену до цього домену. Зонна передача реєструє всі системи і *IP*-адреси домену, відомі *DNS*. Отримавши цей список, зловмисник запускає інструментальні засоби типу *Nmap* для визначення операційної системи потенційного об'єкта атаки. Приховане сканування виявить служби, що виконуються в системі, і ці дані використовуються для реальних атак.

Попереднє дослідження не обмежується збиранням *Інтернет*-адрес. *Wardialing* ("розвідка по телефону") – ще один метод, використовуваний зловмисниками для виявлення потенційних жертв і визначення систем, що мають модем і які відповідають на вхідні дзвінки. За допомогою комп'ютера зловмисник протягом однієї ночі телефонує на тисячі телефонних номерів, знайдених в модемній лінії. Сучасні програмні засоби здатні розрізнити модем і факс. Після виявлення модемів, що відповіли, зловмисник звертається до кожного з них, визначаючи працюючі програми. За допомогою програми *PC Anywhere* зловмисник захоплює управління комп'ютером, що відповів.

Швидке розповсюдження бездротових мереж в організаціях і у домашніх користувачів також дозволяє провести хакерську "розвідку". Новий термін "*wardriving*" ("розвідка на автомобілі") означає, що зловмисник роз'їжджає по місту з комп'ютером і адаптером бездротової мережі, виявляючи точки входу бездротових мереж. При цьому використовується пристрій типу *GPS* (*Global Positioning System* – глобальна система навігації і визначення положення) для запису координат таких точок. Іноді подібна розвідка виконується разом з "*warchalking*". Зловмисник орієнтується по крейдяних відмітках на тротуарах або стінах будівель, які показують, що в цьому місці знаходиться відкрита бездротова.

Після виявлення бездротової мережі зловмисник скористається виходом в *Інтернет* для атаки інших сайтів. Такий спосіб атаки відмінно маскує зловмисника, адже помилковий слід веде до бездротової мережі організації. Навіть у разі виявлення присутності зловмисника з'ясувати

його реальне місце розташування дуже важко.

У загальному випадку зловмисник, що використовує методи ненаправлених атак, має в своєму розпорядженні один або декілька (не дуже багато) експлойтів. За допомогою попередньої розвідки він спробує знайти системи, вразливі до цих експлойтів. Більшість зловмисників, відшукавши систему, спробують зламати її "в один прийом". Більш досвідчені зломщики за допомогою спеціальних засобів сканування знаходять декілька вразливих систем, а потім створюють сценарії атаки, спрямованої проти всіх систем одночасно.

Після зламу системи зловмисник зазвичай поміщає в неї "чорний хід", через який він входитиме в систему надалі. Часто "чорні ходи" використовуються разом з інструментом *rootkit*, що включає версію системи з кодом "троянського коня", який дозволяє приховати присутність зловмисника. Деякі зловмисники закривають вразливе місце, через яке вони проникли всередину, щоб ніхто більше не міг управляти "їх системою". Зловмисники копіюють файли з паролями інших систем, щоб попрацювати на дозвіллі над їх розкриттям, завантажують сніфер для захоплення паролів. Після зламу система використовується для атаки або для попереднього зондування.

Наприклад, в кінці червня 1999 р. багато систем піддалися атаці через *Інтернет* і були успішно зламані. Напад виглядав автоматизованим, оскільки зламування систем відбувалося протягом короткого проміжку часу. Дослідження і аналіз систем показали, що зловмисник застосував для проникнення засіб *RPC Tooltalk*, що викликає переповнювання буфера. Після входу в систему зловмисник запускав сценарій, який виконував такі дії:

- закривав вразливе місце, через яке зловмисник проник в систему;
- завантажував "чорний хід" у файл *inetd*, щоб зловмисник міг повертатися в систему;
- запускав в системі сніфер паролів.

В процесі роботи група дослідників отримала сценарії, які виглядали так, ніби відправлені від системи зловмисника. Але насправді вони працювали на зламаній системі, даючи зловмисникові можливість автоматизованого повернення в кожну розкриту систему і витягання файлів журналу сніфера. Ці файли включали ідентифікатори користувачів і їх паролі для кожної системи в локальній мережі.

Такий тип сценаріїв зустрічається все частіше і частіше. Крім того, поява черв'яків, які діють аналогічним чином і повертають звіт своєму розробникові, показує, що ця атака не була унікальною.

1.5 Виявлення методів направлених атак

Зловмисник, що використовує методи направлених атак, намагається проникнути в конкретну організацію або завдати їй збитків. Мотивацією його дій є прагнення отримати від організації інформацію певного типу.

Він прагне заподіяти шкоду декількома способами, використовуючи для цього спрямовані *DoS*-атаки. Рівень майстерності таких зловмисників вищий, ніж у тих зловмисників, які не мають певної мети.

Вибір об'єкта атаки зазвичай обґрунтований – це інформація, що викликає інтерес у зловмисника. Зловмисника може найняти стороння організація для отримання деяких відомостей. Незалежно від причини, об'єктом атаки стає конкретна організація.

У направлених атаках проводиться фізична розвідка, а також попереднє дослідження адрес, телефонних номерів, системи, сфери діяльності.

В процесі попереднього дослідження адрес виявляється адресний простір організації. Цю інформацію можна знайти в багатьох місцях. Насамперед, служба *DNS* дозволяє визначити адреси веб-серверів організації: адресу головного *DNS*-сервера в домені і адресу поштового сервера. Відшукати потрібні адреси можна за допомогою Американського реєстру номерів *Інтернет* (*ARIN*). У *ARIN* можливий пошук за іменем для знаходження адресних блоків, призначених даній організації.

Додаткові доменні імена, призначені організації, є в *Network Solutions* (тепер частина *VeriSign* – http://www.networksolutions.com/). Для кожного знайденого домену за допомогою служби *DNS* визначаються додатковий веб-сервер, поштовий сервер та діапазон адрес. Пошук цієї інформації не привертає уваги цільової системи.

Багато інформації про використовувані адреси дасть зонна передача від головного *DNS*-сервера домену. Якщо сервер дозволяє здійснювати таку передачу, то в результаті можливе отримання списку всіх відомих йому систем домену. Це дуже цінна інформація, але такий підхід може звернути на себе увагу цільової системи. Правильно налаштовані *DNS*-сервери обмежують зонну передачу. В цьому випадку спроба отримання інформації заноситься в журнал подій і виявляється адміністратором

Використовуючи всі вищеперераховані способи, зловмисник отримує список доменів організації, адреси всіх веб-серверів, поштових серверів і головних серверів, список діапазонів адрес і, потенційно, список всіх використовуваних адрес. Велику частину цієї інформації він знайде, не вступаючи в безпосередній контакт з організацією-жертвою.

1.5.1 Попереднє дослідження телефонних номерів та бездротових мереж

Попереднє дослідження телефонних номерів виконати складніше, ніж відшукати мережеві адреси. Дізнатися головний номер організації можна в довідковій або на веб-сайті, оскільки організації публікують там свої контактні телефони і номери факсів.

Після отримання телефонних номерів зловмисник шукає працюючі модеми, скориставшись програмою типу "*wardialer*". Приблизно визначивши блок телефонних номерів, використовуваних організацією, він починає

дозвон за цими номерами. Проте така діяльність не залишиться непоміченою. Тому зловмисник постарається виконати це в неробочий час або у вихідні дні, щоб зменшити ймовірність виявлення.

Ускладнює роботу та обставина, що він не знає номера напевно. В результаті у нього на руках можуть опинитися модемні підключення інших організацій, які в даний момент йому не дуже-то потрібні.

Врешті-решт, зловмисник отримає список номерів з модемом, що відповідає. Можливо, він стане в нагоді йому, а можливо, і ні. Зловмисникові належить виконати велику роботу для збору необхідної інформації.

Для дослідження бездротових мереж зловмисник перевірить навколишній район (автомобільні стоянки, інші поверхи будівлі, вулицю) на предмет наявності бездротових мереж. Цю розвідку він виконає без особливих зусиль, прогулюючись навколо будівлі або проїжджаючи на автомобілі. В більшості випадків його спроби підключення до бездротової мережі не будуть зареєстровані

Виконуючи таке дослідження, зловмисник повинен фізично знаходитися поряд з цільовим об'єктом.

1.5.2 Попереднє дослідження системи

Попереднє дослідження систем являє потенційну небезпеку для зловмисника, але не з погляду затримання і арешту, а з погляду залучення уваги. В процесі збору даних зловмисник визначає використовуване обладнання, операційні системи і їх вразливі місця.

Зловмисник застосовує розгорнену відправку *ping*-пакетів, приховане сканування або сканування портів. Якщо він хоче залишитися "в тіні", то виконуватиме все дуже повільно – один *ping*-пакет за однією адресою приблизно кожну годину. Така діяльність залишиться непоміченою для більшості адміністраторів.

Сканування для визначення операційних систем приховати важче, оскільки сигнатури пакетів більшості інструментальних засобів добре відомі, і системи виявлення вторгнень (*Intrusion Detection Systems, IDS*) з великою мірою ймовірності виявлять ці спроби. Зловмисник може відмовитися від використання відомих інструментів і застосує приховане сканування портів. Якщо система відповідає через порт 139 (*NETBIOS RPC*), то це, ймовірно, *Windows*, якщо через порт 111 (*Sun RPC/portmapper*) – те це система *Unix*. Поштові системи і веб-сервери виявляються через підключення до певних портів (25 – для пошти, 80 – для веб-сервера) та дослідження відповіді. В більшості випадків можна дізнатися тип використовуваного програмного забезпечення і, отже, операційну систему. Такі підключення виглядають цілком легальними, не привертають уваги адміністраторів або систем виявлення вторгнень *IDS*.

Виявлення вразливих місць являє для зловмисника серйозну небезпеку. Це можна зробити, здійснюючи атаки або досліджуючи систему

на наявність уразливості. Одним із способів є перевірка номера версії популярного програмного забезпечення, наприклад, поштового сервера або *DNS*-сервера, яка і підкаже відомі вразливі місця.

Якщо зловмисник скористається сканером вразливих місць, то він з великою вірогідністю викличе сигнал тривоги в системі *IDS*. Один сканер допоможе зловмисникові відшукати єдину "діру", інший виявить більшу кількість вразливих місць. Незалежно від використовуваного інструменту, зловмисник збере потрібну інформацію, але, швидше за все, його присутність буде відмічена.

1.5.3 Попереднє дослідження сфери діяльності та фізичні методи збору даних

Розуміння сфери діяльності організації дуже важливе для зловмисника. Він повинен знати, як використовуються комп'ютерні системи, де розміщена цінна інформація і апаратура. Ці знання допоможуть йому визначити місце розташування вірогідної мети. Наприклад, якщо сайт електронної комерції замість обробки транзакцій власників кредитних карт відправляє покупців на сайт банку, це означає, що на цільовій системі не зберігаються номери кредитних карт.

При проведенні попереднього дослідження зловмисник постарається з'ясувати, яким способом можна максимально нашкодити системі.

Частиною бізнес-моделі будь-якої організації є розміщення службовців і порядок здійснення ними своїх функцій. Організації, розташовані в одному приміщенні, здатні забезпечити периметр безпеки навколо всіх важливих систем. У організаціях з великою кількістю підрозділів, зв'язаних через *Інтернет* або виділені канали, може бути надійно захищена основна мережа, але слабо – віддалені офіси. Вразливими стають організації, що дозволяють службовцям виконувати віддалене підключення. В цьому випадку домашні комп'ютери співробітників використовують віртуальні приватні мережі для підключення до внутрішньої мережі організації. Найпростіпим способом отримання доступу в організацію в цьому випадку стане зламування однієї з домашніх систем.

І останнім етапом розвідки є збір інформації про службовців. Багато організацій розміщують інформацію про керівників на своєму веб-сайті. Така інформація являє велику цінність, якщо зловмисник задумає скористатися методами соціального інжинірингу. Додаткову інформацію дасть пошук в *Інтернет* за іменем домену організації. У такий спосіб можна отримати адреси електронної пошти службовців, новин, що розміщують в *Інтернет*, або список поштових адрес. Часто в адресах електронної пошти співробітників містяться ідентифікатори користувачів.

Фізичні методи збору даних в основному використовуються в направлених атаках. Часто вони дозволяють дістати доступ до потрібної інформації або комп'ютера без зламу системи безпеки організації.

Зловмисник веде спостереження за будівлею, в якій розміщується організація. Він вивчає компоненти фізичної безпеки: пристрої контролю доступу, камери спостереження і службу охорони. Він спостерігає за тим, як входять відвідувачі, як виходять службовці під час перерви. Таке спостереження дозволяє виявити слабкі сторони системи фізичної безпеки, які можна використовувати для проникнення в будівлю.

Зловмисник слідкує і за тим, як поводяться із сміттям і викинутими документами. Якщо все це складається в сміттєвий бак позаду будівлі, то він може вночі поритися в ньому і знайти потрібну інформацію.

1.5.4 Методи атак

Маючи на руках всю необхідну інформацію про об'єкт атаки, зловмисник вибере найбільш відповідний спосіб з мінімальним ризиком виявлення. Запам'ятаєте, що зловмисник, який здійснює направлену атаку, зацікавлений залишитися в тіні. Він навряд чи вибере метод атаки, який включить систему тривоги. Матимемо це на увазі при вивченні електронних і фізичних методів атак.

Зловмисник провів успішну розвідку і виявив всі зовнішні системи і всі підключення до внутрішніх систем. Під час збору даних про організацію він визначив вразливі місця систем. Вибір певної вразливості повинен бути досить ретельний, оскільки об'єкт атаки може мати системи виявлення вторгнення. Використання відомих методів атак приведе в дію таку систему і викличе у відповідь дії.

Зловмисник спробує приховати атаку від *IDS*, розбиваючи її на декілька пакетів. Але він ніколи не буде упевнений, що атака пройшла непоміченою. Тому в разі успішного завершення атаки він зробить так, щоб стан системи виглядав як завжди. Зловмисник не почне знищувати файли журналів подій, оскільки це відразу приверне увагу адміністратора. Замість цього він знищить записи в журналі, що видають його присутність. Увійшовши до системи, зловмисник встановить "чорний хід" для подальших проникнень в систему.

Якщо зловмисник вирішить атакувати за допомогою дзвінка по телефону, він пошукає видалений доступ з паролем, який легко вгадується, або зовсім без пароля. Його першочерговими цілями стануть системи з видаленим управлінням або системи адміністратора. Він атакує їх в неробочий час, щоб запобігти виявленню атаки службовцями.

Якщо зловмисник знайшов вразливу домашню систему службовця, він атакуватиме її безпосередньо або відправить туди вірус або "троянського коня". Подібна програма потрапляє на комп'ютер у вигляді вкладення в повідомлення електронної пошти, яке самостійно виконується і встанолюється при відкриванні вкладення. Такі програми особливо ефективні, якщо комп'ютер працює під управлінням системи *Windows*.

Соціальний інжиніринг – найбезпечніший метод фізичної атаки, за

допомогою якого можна проникнути в систему. Ключовий момент такої атаки – маленька брехня. Наприклад, зловмисник подзвонить секретареві в приймальні і дізнається номер служби підтримки. Потім він зв'яжеться з віддаленим офісом і під виглядом секретаря довідається про будь-якого службовця. Наступний дзвінок – в службу підтримки – він зробить від його імені: попросить номер телефону для локального дзвінка або скаже, що забув пароль. Здобута інформація дозволить зловмисникові увійти до системи з легальним *ID* і паролем користувача.

Найнебезпечнішим типом фізичної атаки є реальне проникнення в організацію. У цьому посібнику не буде описуватись злам приміщення, хоча серйозний зловмисник зважиться і на це. Опинившись усередині, він підключить свій переносний комп'ютер до мережі. У багатьох компаніях недостатньо контролюються внутрішні мережеві підключення, тому у розпорядженні зловмисника опиниться вся мережа. Якщо службовці не навчені докладати про сторонніх в офісі, у зловмисника буде маса часу для пошуку потрібної інформації.

Зловмисник використовуватиме зламану систему в своїх цілях, прагнучи приховати сліди своєї присутності настільки ретельно, наскільки можливо. Такі зловмисники не хваляться своїми перемогами. Зламана система стане для нього стартовим майданчиком для проникнення в більш засекречені внутрішні системи. Всі дії виконуватимуться максимально приховано, щоб не привернути уваги адміністраторів.

2 ЗАХИСТ НА РІВНІ КОМУНІКАЦІЙНИХ ПРОТОКОЛІВ

2.1 Введення

Сімейство протоколів *TCP/IP* працює на будь-яких моделях комп'ютерів, зроблених різними виробниками комп'ютерної техніки та працюючих під керуванням різних операційних систем. За допомогою протоколів *TCP/IP* можна об'єднати практично будь-які комп'ютери.

Сьогоднішні реалізації протоколу *TCP/IP* дуже далекі від того, як він замислювався спочатку. Наприкінці 60-х років почався дослідницький проект фінансований урядом США з розробки мережі пакетної комутації, а в 90-х роках результати цих досліджень перетворилися в найбільш широко використовувану форму мережевої взаємодії між комп'ютерами. На сьогоднішній день це дійсно відкрита система, а саме, сімейство протоколів і велика кількість безкоштовних реалізацій (або досить дешевих). Вони становлять основу того, що називається словом *Інтернет*.

Мережеві протоколи звичайно розробляються по рівнях, причому кожен рівень відповідає за власну фазу комунікацій. Сімейства протоколів, таких як *TCP/IP*, це комбінації різних протоколів на різних рівнях. Стек протоколів *TCP/IP* складається із чотирьох рівнів, як показано на рисунку 2.1. Кожен рівень несе власне функціональне навантаження.

Прикладний	*Telnet, FTP, e-mail* і т.д.
Транспортний	*TCP, UDP*
Мережевий	*IP, ICMP, IGMP*
Канальний	Драйвер пристрою та інтерфейсна плата

Рисунок 2.1 – Чотири рівні протоколів TCP/IP

Канальний рівень *(link layer)*. Ще його називають рівнем мережевого інтерфейсу. Звичайно містить у собі драйвер пристрою в операційній системі та відповідну мережеву плату комп'ютера. Разом вони забезпечують апаратну підтримку фізичного з'єднання з мережею (з кабелем або з іншим використовуваним середовищем передачі).

Мережевий рівень *(network layer)*, іноді називають рівнем міжмережевої взаємодії, відповідає за передачу пакетів по мережі. Маршрутизація пакетів здійснюється саме на цьому рівні. *IP*-протокол *(Internet Protocol* – протокол *Інтернет), ICMP (Internet Control Message Protocol* – протокол керування повідомленнями *Інтернет)* та *IGMP (Internet Group Management Protocol* – протокол керування групами *Інтернет*) забезпечують мережевий рівень у сімействі протоколів *TCP/IP*.

Транспортний рівень *(transport layer)* відповідає за передавання потоку даних між двома комп'ютерами й забезпечує роботу прикладного

рівня. У сімействі протоколів *TCP/IP* існує два транспортних протоколи: *TCP (Transmission Control Protocol)* та *UDP (User Datagram Protocol).* Протокол *TCP* здійснює надійну передачу даних між двома комп'ютерами. Він забезпечує розподіл даних, що передаються від одного додатка до іншого, на пакети, які мають розмір, що використовується для мережевого рівня розміру, підтвердження прийнятих пакетів, установлення пауз, протягом яких повинне прийти підтвердження на пакет тощо. Оскільки надійність передачі даних гарантується на транспортному рівні, на прикладному рівні ці деталі ігноруються. Протокол *UDP* надає більше простий сервіс для прикладного рівня. Він просто відсилає пакети, які називаються дейтаграмами (*datagram*) від одного комп'ютера до іншого. При цьому немає ніякої гарантії, що дейтаграма дійде до пункту призначення. За надійність передачі даних, при використанні дейтаграм відповідає прикладний рівень. Для кожного транспортного протоколу існують різні додатки, які їх використовують.

Прикладний рівень *(application layer)* визначає деталі кожного конкретного додатка. Існує кілька розповсюджених додатків *TCP/IP,* які присутні практично в кожній реалізації:

- *Telnet* – протокол віддаленого терміналу.
- *FTP (File Transfer Protocol)* – протокол передачі файлів.
- *SMTP (Simple Mail Transfer Protocol)* – простий протокол передачі електронної пошти.
- *SNMP (Simple Network Management Protocol)* – простий протокол керування мережею.

У сімействі протоколів *TCP/IP* мережевий рівень – *IP.* Він надає ненадійний сервіс. Це означає, що в процесі своєї роботи протокол передає пакет від джерела до пункту призначення, однак не надає ніяких гарантій того, що пакет дійде за призначенням. Протокол *TCP,* з іншого боку, надає надійний транспортний рівень, що користується ненадійним сервісом *IP.* Щоб забезпечити подібний сервіс, *TCP* виставляє паузи та здійснює повторні передачі, відсилає й приймає підтвердження тощо. У дійсності сімейство протоколів *TCP/IP* поєднує значно більше протоколів. Протоколи *TCP* та *UDP* – два основних протоколи транспортного рівня. Обидва використовують *IP* як транспортний.

2.2 Протокол *IPv4*

Кожен інтерфейс в об'єднаній мережі повинен мати унікальну *IP*-адресу. Адреси *IPv4* – це 32-бітові числа. Існує певна структура адреси *Інтернет.* На рисунку 2.2 показано 5 класів адрес *Інтернет.*

Ці 32-бітові адреси звичайно записуються як 4 десяткові числа, по одному на кожен байт адреси. Така форма запису називається "десятковим записом із точками" *(dotted-decimal).* Наприклад, адреса мережі класу B може бути записана як 140.252.13.33.

Клас	Діапазон
A	0.0. 0.0 – 127. 255.255.255
B	128.0. 0.0 – 191. 255.255.255
C	192.0. 0.0 – 223. 255.255.255
D	224.0. 0.0 – 239. 255.255.255
E	240.0. 0.0 – 247. 255.255.255

Рисунок 2.2 – Діапазони IPv4 адрес у різних класах мереж

Хости з декількома інтерфейсами мають декілька адрес *IP*: по одній на кожен інтерфейс. Оскільки кожен інтерфейс, підключений до мережі, повинен мати унікальну адресу, виникає питання розподілу *IP*-адрес у глобальній мережі *Інтернет*. Цим займається мережевий інформаційний центр *(Internet Network Information Center – InterNIC)*. Він призначає тільки мережеві ідентифікатори *(ID)*. Призначенням ідентифікаторів хостів у мережі займаються системні адміністратори.

Протокол *IP* – це основний інструмент сімейства протоколів *TCP/IP*. Протоколи *TCP, UDP, ICMP* та *IGMP* передають свої дані як *IP*-дейтаграми. При цьому необхідно враховувати, що протокол *IP* ненадійний, оскільки надає сервіс доставки дейтаграм без з'єднання.

Під словом ненадійний мається на увазі те, що не існує гарантії того, що *IP*-дейтаграма успішно досягне пункту призначення. Однак протокол *IP* надає певний сервіс обробки деяких подій. Коли що-небудь іде не так як хотілося б, наприклад, тимчасове переповнення буфера маршрутизатора, *IP* застосовує простий алгоритм обробки помилок: він відкидає дейтаграму та намагається послати *ICMP*-повідомлення відправникові. Необхідна надійність повинна бути забезпечена верхніми рівнями (наприклад *TCP*).

Термін «без з'єднання» *(connectionless)* означає, що протокол *IP* не містить ніякої інформації про передачу дейтаграм. Кожна дейтаграма обробляється незалежно від інших. Це також означає, що може бути доставлена зіпсована дейтаграма. Якщо джерело відправляє дві послідовні дейтаграми (перша А, друга В) в один і той самий пункт призначення, кожна з них маршрутизується незалежно й може пройти по різних маршрутах. Дейтаграма В може прибути раніше ніж *А*.

Стандартний розмір заголовка *IPv4* становить 20 байт, якщо немає опцій. Старший значущий біт має номер 0 (ліворуч), а молодший значущий біт з 32-х біт має номер 31-го байта. Значення передаються в такому порядку: спочатку біти 0 – 7, потім біти 8 – 15, потім 16 – 23 й, нарешті, 24 – 31. Такий порядок руху байтів називається *big endian* (метод зберігання або передачі даних, при якому старший значущий біт або байт стоїть першим) і обов'язковий для всіх двійкових цілих чисел в *TCP* заголовках

при їхній передачі по мережі. Машини, які зберігають двійкові дані в інших форматах, як наприклад у форматі *little endian* (метод зберігання або передачі даних, при якому молодший значущий біт або байт стоїть першим), повинні конвертувати значення заголовків у відповідному порядку мережевих байтів перед передачею даних.

Структура пакета *IPv4* показана на рис. 2.3.

Біти 0-3	**4-7**	**8-15**	**16-18**	**19-23**	**24-31**
Версія	HLEN	Тип обслуговування	Загальна довжина		
Ідентифікація			Прапорець	Зміщення фрагментації	
Час життя		Протокол	Контрольна сума заголовку		
IP-адреса відправника					
IP-адреса одержувача					
Опції					Додаток
Дані (65535 мінус заголовок)					
...					

Рисунок 2.3 – Структура дейтаграми протоколу IPv4

- **Версія (Version)** – 4-бітове поле, що описує використовувану версію протоколу *IP*. Всі пристрої зобов'язані використовувати протокол *IP* однієї версії, пристрій, що використовує іншу версію буде відкидати пакети.
- **Довжина *IP*-заголовку** (*IP header Length – HLEN*) – 4-бітове поле, що описує довжину заголовка пакета в 32-бітових блоках. Це значення – повна довжина заголовка з врахуванням двох полів змінної довжини. Оскільки це 4-бітове поле, воно обмежує розмір заголовка в 60 байт. Звичайна величина в цьому полі (тобто, якщо відсутні опції) – 5.
- **Тип обслуговування** (*Type of Service – TOS*) – 8-бітове поле, що вказує на ступінь важливості інформації, яка присвоєна протоколом верхнього рівня.
- **Загальна довжина** (*Total Length*) – 16-бітове поле, що описує довжину пакета в байтах, із заголовком і даними включно. Для того щоб вирахувати довжину блока даних, потрібно від повної довжини відняти значення поля *HLEN*. Завдяки цьому полю та полю довжини заголовка, можна визначити з якого місця починаються дані в дейтаграмі та їхню довжину. Оскільки це поле складається з 16 біт, максимальний розмір дейтаграми становить 65535 байт. (Зверніть увагу, що *Hyperchannel* має *MTU*, рівний 65535. У дійсності це не *MTU* – тут використовується максимально можливий розмір дейтаграми). Це поле змінюється в момент фрагментації та повторного збирання дейтаграми.
- **Ідентифікація** (*Identification*) – 16-бітове поле, що зберігає ціле число, яке описує даний пакет. Це число являє собою послідовний номер.

- **Прапорці** (*Flags*) – 3-бітове поле, в якому два молодших біти контролюють фрагментацію пакетів. Перший біт визначає чи був пакет фрагментований, а другий чи цей пакет останній в серії фрагментів.
- **Зміщення фрагментації** (*Fragment Offset*) – 13-бітове поле, що допомагає зібрати разом фрагменти пакетів. Це поле дозволяє використовувати 16 бітів в сумі для прапорців фрагментації.
- **Час життя** (*Time-to-Live – TTL*) – 8-бітове поле – лічильник, в якому зберігається послідовно зменшуване значення кількості пройдених вузлів (маршрутизаторів, яких в цьому випадку називають хопами (*hops*)) на шляху до місця призначення. У випадку, коли лічильник пройдених хопів дорівнюватиме нулю – пакет буде відкинуто, таким чином попереджається нескінченна циклічна пересилка пакетів.
- **Протокол** (*Protocol*) – 8-бітове поле, що вказує на те, який протокол верхнього рівня отримає пакет, після завершення обробки протоколом *IP*. Наприклад *TCP* або *UDP*.
- **Контрольна сума заголовка** (*Header Checksum*) – 16-бітове поле, що допомагає перевірити цілісність заголовку пакета. Щоб розрахувати контрольну суму *IP* для вихідної дейтаграми, поле контрольної суми спочатку встановлюється в 0. Потім розраховується 16-бітова сума з порозрядним доповненням *(One's complement* – порозрядне доповнення до двійкової системи. Заголовок цілком сприймається як послідовність 16-бітових слів). 16-бітове порозрядне доповнення цієї суми зберігається в полі контрольної суми. Коли дейтаграма приймається, обчислюється 16-бітова сума з порозрядним доповненням. Контрольна сума, розрахована приймачем, містить у собі контрольну суму, збережену відправником. Контрольна сума приймача складається з бітів рівних 1, якщо в заголовку нічого не було змінено при передачі. Якщо в результаті всі біти не одиничні (помилка контрольної суми), *IP* відкидає прийняту дейтаграму. Повідомлення про помилку не генерується. Тепер завдання верхніх рівнів певним чином визначити, що дейтаграма відсутня, і забезпечити повторну передачу. *ICMP, IGMP, UDP* та *TCP* використовують такий же алгоритм розрахунку контрольної суми.
- **IP-адреса відправника** (*Source IP address*) – 32-бітове поле, що зберігає *IP*-адресу вузла-відправника.
- **IP-адреса одержувача** (*Destination IP address*) – 32-бітове поле, що зберігає адресу вузла призначення.
- **Опції** (*Options*) – поле змінної довжини, що дозволяє протоколу *IP* реалізувати підтримку різних опцій, зокрема засобів безпеки.
- **Додаток** (*Padding*) – поле, що використовується для вставки додаткових нулів, для гарантування кратності *IP*-заголовку 32 бітам.
- **Дані** (*Data*) – поле змінної довжини (максимально 64 Кбіт), яке зберігає інформацію для верхніх рівнів.

IP-пакет складається з даних протоколу верхнього рівня і заголовка, що має описану вище структуру. Хоча основною частиною заголовка є адреси відправника і призначення, саме інші частини заголовка роблять протокол таким надійним і гнучким. Інформація, що зберігається в полях заголовка задає дані пакета і призначена для протоколів верхніх рівнів.

2.3 Протокол *IPv6*

Протокол *IPv6* – нова версія *IP*-протоколу версії 6. Розробка протоколу *IPv6* почалася в 1992 році, а з 2003 р. його підтримка забезпечується виробниками більшості телекомунікаційного устаткування (корпоративного рівня). *IPv6* є новим кроком у розвитку *Інтернету*. Цей протокол розроблений з урахуванням зростаючих вимог до Глобальної мережі. При нинішніх темпах розвитку мережі адреси за умови використання старого протоколу закінчаться до 2017 року. Протокол був розроблений *IETF*.

5 лютого 2008 року вночі організація *ICANN*, яка наглядає за використанням протоколів *Інтернет*, почала додавати в *DNS*-сервери записи, що містять адреси у форматі протоколу *IPv6*. Це поклало початок переходу з нинішнього протоколу *IPv4* на сучасніший *IPv6*.

Документом, що фіксує появу *IPv6*, є специфікація *RFC 1752*.

Різниця між *IPv4* та *IPv6* полягає в тому, що раніше на *IP*-адресу виділяли 4 байти (32 біти), що відповідає чотириблоковій стандартній на сьогодні *IP*-адресі, а протокол *IPv6* виділяє на адресу 16 байт (128 біт).

У специфікації *RFC 1726* поданий набір функцій, основними серед них є:

- **масштабованість**: ідентифікація та визначення адрес як мінімум 10^{12} кінцевих систем і 10^9 індивідуальних мереж;
- **топологічна гнучкість**: архітектура маршрутизації і протокол повинні працювати в мережах з різною топологією;
- **спадкосмність**: забезпечення чіткого плану переходу від існуючої версії *IPv4*;
- **незалежність від середовища передачі**: робота серед множини мереж з різними середовищами передачі даних із швидкостями до сотень гігабіт в секунду;
- **автоматична конфігурація хостів і маршрутизаторів**;
- **безпека на мережевому рівні**;
- **мобільність**: забезпечення роботи з мобільними користувачами, мережами та міжмережевими системами;
- **розширюваність**: можливість подальшого розвитку відповідно до нових потреб.

В результаті реалізації заявлених функцій найважливіші інновації *IPv6* полягають в тому, що:

- спрощений стандартний заголовок *IP*-пакета;

- змінено подання необов'язкових полів заголовка;
- розширений адресний простір;
- покращена підтримка ієрархічної адресації, агрегації маршрутів і автоматичної конфігурації адрес;
- застосовано механізми автентифікації і шифрування на рівні *IP*-пакетів;
- введені мітки потоків даних.

При цьому в *IPv6* всі зміни планувалися так, щоб мінімізувати зміни на інших рівнях протокольного стека *TCP/IP*. В результаті розмір *IP*-адреси збільшений до 128 біт (16 байт). Навіть з урахуванням неефективності використання адресного простору, оборотною стороною ефективної маршрутизації та автоматичної конфігурації, достатньо для забезпечення об'єднання мільярда мереж, як того вимагали документи *IETF*. Забезпечена можливість простої і гнучкої автоматичної конфігурації адрес для мереж довільного масштабу і складності. Протокол *IPv6* залишився розширюваним протоколом, причому поля розширень (додаткові заголовки) можуть додаватися без зниження ефективності маршрутизації.

Введення в протоколі *IPv6* поля *Мітка потоку* дозволяє значно спростити процедуру маршрутизації однорідного потоку пакетів. Потік – це послідовність пакетів, що посилаються відправником певному адресатові, при цьому передбачається, що всі пакети даного потоку повинні бути певним чином оброблені. Характер даної обробки задається додатковими заголовками.

Існують різні типи адрес *IPv6*: одноадресні (*Unicast*), групові (*Anycast*) і багатоадресні (*Multicast*).

Адреси типу *Unicast* добре всім відомі. Пакет, на таку адресу, досягає того інтерфейсу, якому цей пакет адресовано.

Адреси типу *Anycast* синтаксично не відрізняються від адрес *Unicast*, але вони адресуються групі інтерфейсів. Пакет, направлений на таку адресу, потрапить в найближчий (згідно з відповідною матрицею маршрутизатора) інтерфейс. Адреси *Anycast* можуть використовуватися тільки маршрутизаторами.

Адреси типу *Multicast* ідентифікують групу інтерфейсів. Пакет, на таку адресу, досягне всіх інтерфейсів, прив'язаних до групи багатоадресного мовлення. Широкомовні адреси *IPv4* виражаються адресами багатоадресного мовлення *IPv6*.

В таблиці 2.1 показані зарезервовані адреси протоколу *IPv6*.

Адреси *IPv6* відображаються як 8 груп шістнадцяткових цифр, розділених двокрапкою. Наприклад,

`7628:0d18:11a3:09d7:1f34:8a2e:07a0:765d.`

Якщо одна або більше груп підряд рівні 0000, то вони можуть бути опущені і замінені на подвійну двокрапку (::).

Таблиця 2.1 – Зарезервовані адреси протоколу *IPv6*

IPv6 адреса	Довжина префікса (біт)	Опис	Примітки
::	128	-	cf. 0.0.0.0 в *IPv4*
::1	128	*loopback* адреса	cf. 127.0.0.1 в *IPv4*
::00:xx:xx:xx:xx	96	вбудований *IPv4*	Нижні 32 біти – це адреса *IPv4*. Також називається *IPv4 сумісною адресою IPv6*
::ff:xx:xx:xx:xx	96	Адреса *IPv6*, відображена на *IPv4*	Нижні 32 біти – адреса *IPv4*. Для хостів, які не підтримують *IPv6*.
fe80:: – feb::	10	*link-local*	*cf. loopback* адреса в *IPv4*
fec0:: – fef::	10	*site-local*	
ff::	8	широкомовний	

Наприклад, `7628:0000:0000:0000:0000:0000:ae21:ad12` може бути скорочена до `7628::ae21:ad12`, або `0000:0000:0000:0000:0000:0000:ae21:ad12` може бути скорочена до `::ae21:ad12`. Скороченню не можуть бути піддані 2 розділених нульових групи через виникнення неоднозначності.

Протокол *IPv6* вирішує потенційну проблему *IP*-адрес за допомогою використання 128-розрядних адрес замість 32-розрядних адрес *IPv4*, завдяки чому адресний простір розширюється в 296 разів. Формат заголовка наведений на рис.2.4.

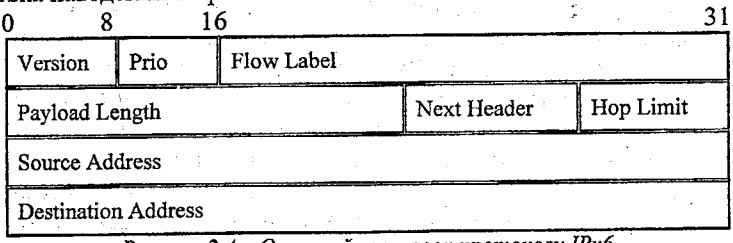


Рисунок 2.4 – Основний заголовок протоколу IPv6

Version (4 біти). *Версія протоколу*. Значення цього поля рівне 6.

Prio (4 біти). *Пріоритет*. Поле пріоритету дозволяє відправникові призначати дейтаграмі певний рівень пріоритету щодо інших пакетів, які відправляються. Можливі 16 значень цього поля розділено на дві категорії: значення поля від 0 до 7 використовується для дейтаграм, які можуть не передаватися при переповненій лінії. Сюди відноситься трафік *TCP*, передача *e-mail*, *FTP*, *NFS*, *Telnet*. Значення поля від 8 до 15 призначені пакетам, які повинні бути відправлені при будь-якому стані (окрім обриву) лінії. Наприклад, пріоритет 8 користувача може призначатися пакетам, які можна відправити в останню чергу при перенавантаженні лінії, а пріоритет 15 – в першу. Останні – це пакети реального часу з відео-, аудіо- та аналогічними даними, які повинні передаватися з постійною швидкістю.

Рекомендовані коди пріоритету для певних видів додатків показані в таблиці 2.2.

Таблиця 2.2 – Коди пріоритету

Код	Призначення
0	Трафік, що не характеризується
1	Заповнювальний трафік (мережеві новини)
2	Неістотний інформаційний трафік (електронна пошта)
3	Резерв
4	Істотний трафік (*FTP*, *HTTP*, *NFS*)
5	Резерв
6	Інтерактивний трафік (*Telnet, X-terminal, SSH*)

Flow Label (24 біти). *Мітка потоку.* Це поле може використовуватися відправником для того, щоб позначати пакети, які вимагають спеціальної обробки мережевими модулями *IPv6*. Хости або шлюзи, що не підтримують цієї опції, повинні встановити це поле в 0 і ігнорувати при обробці пакета.

Потоком є послідовність пакетів, що відправляються певному одержувачу (або групі одержувачів), на шляху до яких пакети повинні пройти спеціальну обробку (наприклад, інформація про проходження певного потоку реєструватиметься). Таких потоків між одними і тими ж хостами може бути декілька, і значення цього поля дозволяє ідентифікувати окремий потік. Якщо значення цього поля – нуль, то вважається, що дейтаграма не належить до жодного потоку. Міткою потоку служить псевдовипадкове число в діапазоні 1 – FFFFFF. Це число також може служити хеш-ключем для шлюзів-обробників певного потоку.

Всі пакети, що належать до одного потоку, повинні відправлятися поодинці і на одну і ту ж адресу призначення і з одним і тим же пріоритетом. Крім того, якщо одна з дейтаграм потоку містить в своєму заголовку будь-який вкладений заголовок або опцію, решта всіх пакетів потоку теж повинна їх містити. Шлюз, який обробляє пакет відхилення складу дейтаграми від інших дейтаграм потоку, генерує помилку потоку і повідомляє про це відправника. Інформація про потік на шлюзі зберігається протягом 6 секунд. Якщо за цей час через шлюз не пройде жодної дейтаграми потоку, ідентифікатор даного потоку звільняється. З іншого боку, хост відправника, у разі перезавантаження (ініціалізації) вузла, не зможе раніше ніж через 6 секунд організувати новий потік.

Payload Length (16 біт). *Довжина даних.* Це довжина даних пакета (у байтах), які слідують за заголовком. Якщо величина цього поля дорівнює 0, то довжина даних дейтаграми більша 65535 і зберігається в полі *Jumbo Payload* (наддовжина) заголовка *Hop-by-Hop Options*.

Поле протоколу *IPv4 Total Length* було перейменоване в протоколі

IPv6 в полі *Payload Length*. Ці два поля схожі між собою, але не тотожні, оскільки поле *Payload Length* містить довжину даних після заголовка, тоді як поле *Total Length* враховує довжину заголовка.

Next Header (8 бітів). *Поле наступного заголовка*. Це поле містить інформацію типу заголовка, який слідує за заголовком *IPv6*.

Ще одне перейменоване і змінене поле *IPv4* — це поле *Protocol*. Воно перетворилося на поле *Next Header* в *IPv6*. У *IPv4* значення поля *Protocol*, наприклад 6 для *TCP* або 17 для *UDP*, визначає, дані якого транспортного протоколу слідують за *IP*-заголовком. У *IPv6* поле *Next Header* дозволяє вставляти додаткові заголовки даними *IP* і *TCP* або *UDP*. Воно також повідомляє тип транспортних даних, наступних за основним або додатковим *IP*-заголовком. Крім того, оскільки поле *Next Header* надає інформацію про наявність додаткових заголовків, наступних за основним, даний механізм виключає необхідність в полі *Internet Header Length*.

Hop Limit (8 бітів). *Поле обмеження пересилок*. Величина цього поля зменшується на 1 при проходженні дейтаграмою шлюзу або хоста. Якщо величина цього поля дорівнює 0, дейтаграма знищується.

Одиниця вимірювання в полі *Time to Live* в *IPv4* — секунди. Проте при проходженні пакета через маршрутизатор, зважаючи на трудність визначення часу очікування в черзі, значення цього поля зменшується реально на 1 секунду. Визнавши ефективність використання числа транзитних маршрутизаторів як спосіб визначення терміну життя пакета, *Png Directorate* замінив поле *Time To Live* на поле *Hop Limit* в *IPv6*.

Source Address (128 бітів). *Адреса відправника*.

Destination Address (128 бітів). *Адреса одержувача*. Якщо в заголовку присутній вкладений заголовок маршрутизації (*Routing header*), це поле може і не бути адресою призначення.

2.4 Система імен доменів

Незважаючи на те, що кожен мережевий інтерфейс комп'ютера має свою власну *IP*-адресу, користувачі звикли працювати з іменами хостів. Існує розподілена світова база даних *TCP/IP*, що називається системою імен доменів *(DNS – Domain Name System)*, яка дозволяє встановити відповідність між *IP*-адресами та іменами хостів. Необхідно бути впевненим, що будь-який додаток може викликати функцію зі стандартної бібліотеки, для того щоб визначити *IP*-адресу (або адреси, що відповідають даному імені хоста). Точно так само ця функція надає можливість здійснити й зворотну процедуру, тобто за заданою *IP*-адресою визначити відповідне ім'я хоста.

2.5 Протокол *TCP*

Протокол управління передачею інформації (*Transmission Control Protocol – TCP*) був розроблений для підтримки інтерактивного зв'язку між

комп'ютерами. Протокол *TCP* забезпечує надійність і достовірність обміну даними між процесами на комп'ютерах, що входять в загальну мережу.

На жаль, протокол *TCP* не пристосований для передачі мультимедійної інформації. Основна причина – відсутність контролю за доставкою. Контроль займає багато часу для передачі чутливої до затримок інформації. Крім того, *TCP* передбачає механізми управління швидкістю передачі з метою уникнення перевантажень мережі. Проте аудіо- та відеодані потребують певних швидкостей передачі, які не можна змінювати довільним чином.

З одного боку протокол *TCP* взаємодіє з прикладним протоколом додатка користувача, а з іншого – з протоколом, що забезпечує "низькорівневі" функції маршрутизації та адресації пакетів, які, як правило, виконує протокол *IP*.

Щоб встановити з'єднання між двома процесами на різних комп'ютерах мережі необхідно знати не тільки *IP*-адреси комп'ютерів, але і номери тих *TCP*-портів (*sockets*), які процеси використовують на цих комп'ютерах. Будь-яке *TCP*-з'єднання в мережі *Інтернет* однозначно ідентифікується двома *IP*-адресами і двома номерами *TCP*-портів.

Протокол *TCP* уміє працювати з пошкодженими, втраченими, дубльованими або такими, що поступили з порушенням порядку проходження пакетами. Це досягається завдяки механізму надання кожному пакету порядкового номера і механізму перевірки отримання пакетів.

Коли протокол *TCP* передає сегмент даних, копія цих даних поміщається в чергу повтору передачі і запускається таймер очікування підтвердження.

2.6 Протокол *ARP*

IP-адреси мають значення тільки в сімействі протоколів *TCP/IP*. Канальні рівні, такі як *Ethernet* або *Token ring*, мають власну схему адресації (в основному 48-бітові адреси); мережеві рівні, у свою чергу, використовують ці канальні рівні. Іншими словами, виникає необхідність установити відповідність між двома різними формами адрес: 32-бітовими *IP*-адресами і будь-яким типом адрес канального рівня. Це можливо за допомогою протоколу визначення адреси *ARP (Address Resolution Protocol)* зворотного протоколу визначення адреси *RARP (Reverse Address Resolution Protocol)*.

ARP надає динамічне зіставлення *IP* адрес і відповідних апаратних адрес. *RARP*, в основному, використовується системами без жорстких дисків (бездискові робочі станції або *X*-термінали), однак тут потрібна ручна конфігурація за участю системного адміністратора.

Ефективність функціонування *ARP* багато в чому залежить від *ARP* кеша *(ARP cache)*, наявного на кожному хості. У кеші містяться *IP*-адреси й відповідні їм апаратні адреси. Стандартний час життя кожного запису в кеші становить до 20 хвилин з моменту створення запису.

3 ЗАХИСТ *IP*

Співтовариство *Інтернет* розробило механізми захисту для цілого ряду програмних додатків, включаючи електронну пошту *(PGP)*, додатки типу клієнт-сервер *(Kerberos)*, додатки доступу до ресурсів *Web (SSL)* тощо. Однак деякі питання захисту не укладаються в рамки протоколу *IP* (*Internet Protocol* – протокол міжмережевої взаємодії). Наприклад, підприємство може захищати свою мережу *TCP/IP*, заборонивши доступ до ненадійних вузлів, шифруючи пакети даних, що залишають мережу підприємства, і вимагаючи автентифікації пакетів, що приходять у мережу ззовні. За допомогою захисту на рівні *IP* підприємство (організація) може забезпечити безпеку не тільки мережевих додатків, що мають свої вбудовані засоби захисту, але й додатків, що не мають таких можливостей.

Захист на рівні *IP* охоплює три сфери безпеки: автентифікацію, конфіденційність та керування ключами. Механізм автентифікації повинний гарантувати, що отриманий пакет був насправді відправлений стороною, зазначеною як джерело повідомлення в заголовку пакета. Крім того, той же механізм повинен гарантувати, що пакет під час передавання не був змінений. Засоби конфіденційності повинні забезпечити можливість шифрування переданих повідомлень, щоб виключити ризик читання таких повідомлень третьою стороною. Засоби керування ключами повинні гарантувати захист процесу обміну ключами.

3.1 Огляд можливостей захисту на рівні *IP*

У 1994 році Рада з архітектури *Інтернет (IAB)* опублікувала звіт, названий "Захист у рамках архітектури *Інтернет*" (документ *RFC* 1636, *"Security in the Internet Architecture"*). Звіт відобразив загальне розуміння того, що *Інтернет* має потребу в більшому і кращому захисті, і визначив області застосування ключів у механізмах захисту. Відзначалася необхідність захисту мережевої інфраструктури від несанкціонованого моніторингу і керування потоками даних, а також захисту обміну даними між користувачами за допомогою засобів автентифікації і шифрування.

Такі вимоги цілком виправдані. Підтвердженням можуть служити дані звітів різноманітних організацій захисту, в тому числі *CERT (Computer Emergency Response Team* – група комп'ютерної "швидкої допомоги"). До найбільш серйозних типів порушень відноситься фальсифікація адрес *IP* і різні форми перехоплення пакетів з даними (у результаті чого порушники одержують передану інформацію, включаючи інформацію автентифікації і зміст баз даних).

У відповідь на цю загрозу *IAB* визнав за необхідне розглядати автентифікацію та шифрування як обов'язкові засоби захисту протоколу *IPv6*. Проте ці засоби можна застосовувати і з протоколом *IPv4*.

3.1.1 Області застосування *IPSec*

Протокол *IPSec* забезпечує захист обміну даними в локальних мережах, корпоративних і відкритих глобальних мережах і в *Інтернет*. Приклади його застосування включають:

- **Захищений доступ до філії організації через** *Інтернет*. Компанія може побудувати захищену приватну віртуальну мережу в рамках мережі *Інтернет* або іншої відкритої глобальної мережі. Це дозволяє використовувати канали відповідної мережі і таким чином скоротити витрати на створення і підтримку приватної мережі.
- **Захищений віддалений доступ через** *Інтернет*. Кінцевий користувач, у системі якого передбачені протоколи захисту, може одержати захищений доступ до мережі компанії. Це скорочує транспортні витрати, які виникають, наприклад, у надомних працівників.
- **Внутрішньомережева і міжмережева взаємодії з партнерами.** Засоби *IPSec* можуть забезпечити захищений зв'язок з іншими організаціями, в склад якого входять механізми автентифікації і конфіденційності, а також механізм обміну ключами.
- **Посилення захисту електронних комерційних операцій.** Навіть якщо деякі додатки *Web* і електронної комерції мають вбудовані протоколи захисту даних, використання *IPSec* підсилює цей захист.

Головною особливістю *IPSec*, яка дозволяє цьому протоколу підтримувати найрізноманітніші програмні додатки, є можливість шифрування і/або автентифікації всього потоку даних на рівні протоколу *IP*. Таким чином, захист може бути забезпечений будь-якому розподіленому додатку, включаючи додатки віддаленої реєстрації, додатки типу клієнт-сервер, електронної пошти, передачі файлів, доступу до ресурсів *Web* тощо.

На рис. 3.1 показаний типовий сценарій використання *IPSec*. Деяка організація підтримує ряд локальних мереж, які знаходяться в різних місцях. У рамках цих локальних мереж трафік *IP* не захищається. Для обміну даними через корпоративну або відкриту зовнішню глобальну мережу використовуються протоколи *IPSec*. Ці протоколи застосовуються пристроями, розміщеними в мережі (наприклад, маршрутизаторами або міжмережевими екранами) і з'єднують локальні мережі з зовнішнім світом. Мережевий пристрій, який використовує протокол *IPSec*, шифрує і ущільнює весь потік даних, який відправляється в глобальну мережу, та дешифрує і розгортає дані, які присилаються ззовні.

Усі виконувані в цьому випадку операції не помітні для робочих станцій і серверів локальної мережі. Захищений обмін даними можливий і з індивідуальними користувачами, які використовують віддалений доступ до мережі через глобальні мережі. Щоб забезпечити захист, робочі станції таких користувачів теж повинні використовувати протоколи *IPSec*.

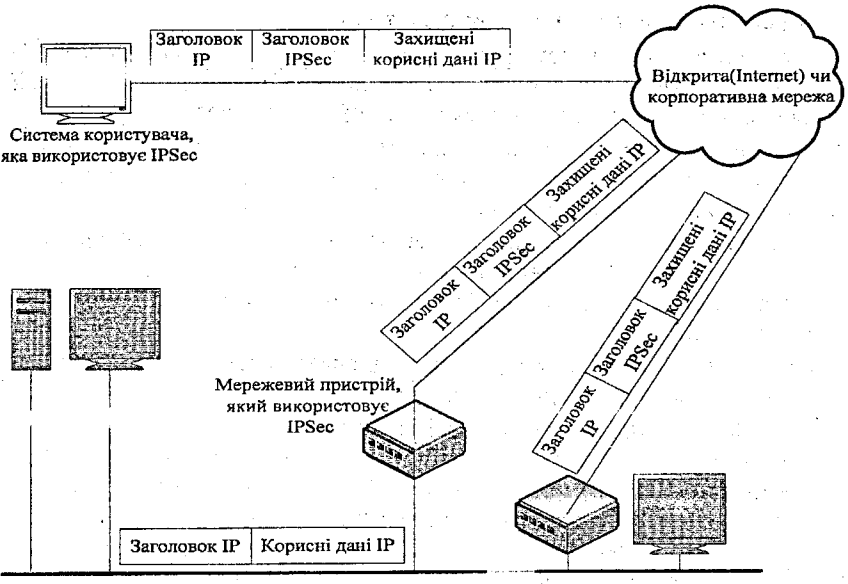


Рисунок 3.1 – Захист на рівні IP

Переваги *IPSec*

Якщо підтримку *IPSec* реалізувати в міжмережевому екрані або маршрутизаторі, це забезпечить надійний захист усього потоку даних, який передається через границю локальної мережі. При цьому потік даних усередині локальної мережі не перевантажується операціями, пов'язаними із захистом.

Засоби *IPSec* у міжмережевому екрані важко обійти, якщо використання протоколу *IP* передбачається для всього потоку даних, а міжмережевий екран є єдиною точкою, що зв'язує мережу організації з *Інтернет*.

Засоби *IPSec* розміщаються на рівні, нижчому за транспортний *(TCP, UDP)*, а тому непомітні для програмних додатків. Отже, немає необхідності змінювати програмне забезпечення в системах користувача або сервера, якщо в міжмережевому екрані або маршрутизаторі реалізується *IPSec*. Навіть якщо *IPSec* реалізується в кінцевих системах, на програмне забезпечення верхнього рівня це не впливає.

Використання *IPSec* може бути непомітним для кінцевого користувача. Таким чином немає необхідності пояснювати користувачеві механізми захисту, видавати інструкції та наполягати на їх нерозголошенні.

При необхідності *IPSec* може забезпечити захист індивідуальним користувачам. Це може знадобитися для осіб, що працюють за територією підприємства, або при створенні захищеної віртуальної підмережі усередині організації для роботи з особливо важливими даними.

3.1.2 Програмні додатки маршрутизації

Крім підтримки кінцевих користувачів та захисту систем і мереж підприємства *IPSec* може брати участь у створенні архітектури маршрутизації при міжмережевій взаємодії. Застосування *IPSec* може гарантувати, що:

- вхідне повідомлення маршрутизатора (тобто повідомлення нового маршрутизатора про його присутність у мережі) дійсно надходить від авторизованого маршрутизатора;
- вхідне повідомлення сусіднього маршрутизатора (тобто маршрутизатора, що намагається встановити стосунки сусідства з маршрутизатором з іншого домену маршрутизації) дійсно надходить від авторизованого маршрутизатора;
- вхідне повідомлення переадресації надходить саме від того маршрутизатора, якому посилався вихідний пакет;
- відновлення маршруту не є фальсифікованим.

Якщо не використовувати засоби захисту, зловмисник може розірвати зв'язок або направити потік даних в обхід по деякому іншому шляху. Для захищених зв'язків між маршрутизаторами, визначених протоколом *IPSec*, повинні підтримуватися протоколи маршрутизації типу *OSPF* (*Open Shortest Path First* – першочергове відкриття найкоротших маршрутів).

3.2 Архітектура захисту на рівні *IP*

Специфікації *IPSec* досить складні. Щоб одержати загальне уявлення про архітектуру *IPSec*, почнемо з обговорення документів, які визначають цей протокол.

3.2.1 Документи *IPSec*

Специфікації *IPSec* визначаються цілим переліком документів. Найбільш важливими з них є:

- *RFC 2401* – огляд архітектури захисту;
- *RFC 2402* – опис розширень автентифікації пакетів *IPv4* і *IPv6;*
- *RFC 2406* – опис розширень шифрування пакетів *IPv4* і *IPv6*;
- *RFC 2408* – специфікації засобів керування ключами.

Підтримка цих можливостей обов'язкова для *IPv6* та можлива для *IPv4*. В обох випадках засоби захисту реалізуються у вигляді заголовків розширень, які розташовані за основним заголовком *IP*. Заголовок розширення автентифікації називають заголовком *AH* (*Authentication Header* – заголовок автентифікації), а заголовок розширення шифрування – заголовком *ESP* (*Encapsulating Security Payload header* – заголовок захищеного корисного вантажу або заголовок захищеного вмісту).

На додаток до чотирьох основних документів робоча група розробки протоколу захисту *IP*, створена *IETF*, опублікувала ще ряд стандартів. Усі документи розділені на сім груп, показаних на рис. 3.2.

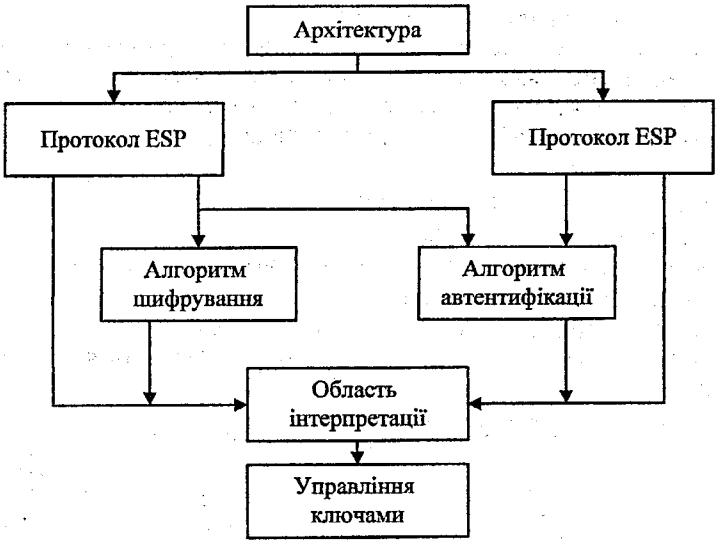


Рисунок 3.2 – Загальна структура документації IPSec

- **Архітектура.** Містить опис загальних принципів, вимог захисту, а також визначення і механізми реалізації технології *IPSec*.
- **Захищений корисний вантаж** *(ESP)*. Опис формату пакета і загальних принципів використання ESP для шифрування пакетів і, якщо потрібно, для автентифікації.
- **Заголовок автентифікації** (*AH*). Опис формату пакета і загальних принципів використання *AH* для автентифікації пакетів.
- **Алгоритм шифрування.** Набір документів, що визначають використання різних алгоритмів шифрування для *ESP*.
- **Алгоритм автентифікації.** Набір документів, що визначають використання різних алгоритмів автентифікації для *AH* і для опції автентифікації *ESP*.
- **Керування ключами.** Документи, що описують схеми керування ключами.
- **Область інтерпретації** (*DOI – Domain of Interpretation*). Містить відомості, необхідні для встановлення відповідності одних документів іншим. Це, зокрема, ідентифікатори перевірених алгоритмів шифрування й автентифікації, а також деякі параметри, наприклад, тривалості життєвого циклу ключів.

3.2.2 Сервіс протоколу *IPSec*

Протокол *IPSec* забезпечує сервіс захисту на рівні протоколу *IP*, дозволяючи системі вибрати необхідні протоколи захисту, визначити алгоритм для відповідного сервісу і задати значення криптографічних

ключів, які необхідні для даного сервісу.

Для захисту використовується два протоколи: протокол автентифікації, зазначений заголовком автентифікації *AH*, та комбінований протокол шифрування / автентифікації, визначений форматом пакета для протоколу *ESP*. У цьому випадку забезпечуються такі види сервісу:

- керування доступом;
- цілісність без встановлення з'єднань;
- автентифікація джерела даних;
- заборона відтворених пакетів (форма цілісності послідовностей);
- конфіденційність (шифрування);
- обмежена конфіденційність транспортного потоку.

У табл. 3.1 показано, які з цих сервісів забезпечуються протоколами *AH* і *ESP*.

Таблиця 3.1 – Сервіс протоколу *IPSec*

	AH	*ESP* (тільки шифрування)	*ESP* (шифрування й автентифікація)
Керування доступом	Так	Так	Так
Цілісність без установлення з'єднань	Так		Так
Автентифікація джерела даних	Так		Так
Відторгнення відтворених пакетів	Так	Так	Так
Конфіденційність		Так	Так
Обмежена конфіденційність транспортного потоку		Так	Так

У протоколі *ESP* існує два варіанти – з використанням і без опції автентифікації. Як протокол *AH*, так і протокол *ESP* мають можливості керування доступом, основані на розподілі криптографічних ключів і керуванні транспортними потоками, пов'язаними з цими протоколами захисту.

3.2.3 Захищені зв'язки

Ключовим об'єктом у механізмах автентифікації та конфіденційності протоколу *IP* є захищений зв'язок (*Security Association*), який є одностороннім з'єднанням між відправником і одержувачем, що застосовує сервіс захисту до транспортного потоку. Якщо потрібне рівноправне з'єднання для двостороннього захищеного обміну, необхідні два захищені зв'язки. Сервіс захисту дає можливість для захищеного зв'язку використовувати або *AH*, або *ESP*, проте в жодному разі не можна використовувати ці протоколи одночасно.

Захищений зв'язок однозначно визначається такими параметрами:

- **Індекс параметрів захисту**. Рядок бітів, який притаманний даній захищеній мережі і має тільки локальне значення. Індекс параметрів захисту передається в заголовках *AH* і *ESP*, щоб приймальна система мала можливість вибрати захищений зв'язок.

- **Адреса IP пункту призначення**. В наш час допускаються тільки односпрямовані адреси – це адреси пункту призначення захищеного зв'язку, які можуть бути системою кінцевого користувача або мережевим об'єктом типу міжмережевого екрана або маршрутизатора.
- **Ідентифікатор протоколу захисту**. Цей ідентифікатор указує, чи є даний зв'язок захищеним зв'язком *AH* або це захищений зв'язок *ESP*.

Таким чином, у будь-якому пакеті *IP* захищений зв'язок однозначно ідентифікується адресою пункту призначення, зазначеною у заголовку *IPv4* або *IPv6*, та індексом параметрів захисту у вкладеному заголовку розширення (*AH* або *ESP*).

3.2.4 Параметри захищеного зв'язку

У кожній реалізації протоколу *IPSec* є номінальна таблиця захищених зв'язків (*Security Association Database*), яка визначає параметри захищених зв'язків. Захищений зв'язок характеризується такими параметрами:

- **Лічильником порядкового номера.** 32-бітове значення, використовуване при генеруванні значень поля чи порядкового номера в заголовках *AH* або *ESP* (потрібний у всіх реалізаціях).
- **Прапорцем переповнення лічильника порядкового номера**. Прапорець, який вказує на переповнення лічильника порядкового номера. В цьому разі генерується повідомлення в журналі реєстрації та припиняється подальша передача пакетів із застосуванням цього захищеного зв'язку (потрібний у всіх реалізаціях).
- **Вікном захисту від відтворення**. Використовується для виявлення відтворених пакетів серед вхідних пакетів *AH* або *ESP*.
- **Інформацією** *AH*. Алгоритм автентифікації, ключі, параметри тривалості життя ключів і інші необхідні параметри, використовувані в рамках *AH*.
- **Інформацією** *ESP*. Алгоритм шифрування та автентифікації, ключі, значення ініціалізації, параметри тривалості життя ключів і інші необхідні параметри, використовувані в рамках *ESP* (потрібний в реалізаціях *ESP*).
- **Тривалістю життя даного захищеного зв'язку**. Інтервал часу або значення лічильника байтів, після досягнення якого захищений зв'язок повинен бути замінений (з новим індексом параметрів захисту) або закінчений з вказанням того, яка саме з цих подій повинна відбутися (потрібний у всіх реалізаціях).
- **Режимом протоколу** *IPSec*. Тунельний, транспортний або заданий груповим символом (потрібний у всіх реалізаціях). (Режими описані нижче).
- **Максимальною одиницею передачі** (*Maximum Transmission Unit – MTU*) **маршруту**. Максимальна одиниця передачі (максимальний розмір пакета, що може бути переданий без фрагментації) для всіх ділянок маршруту і змінних часу існування (потрібний у всіх реалізаціях).

- **Механізмом керування ключами,** пов'язаним з механізмами автентифікації і конфіденційності тільки через індекс параметрів захисту. Таким чином, автентифікація і конфіденційність визначаються в залежності від конкретного механізму керування ключами.

3.2.5 Селектори захищеного зв'язку

Протокол *IPSec* забезпечує користувачеві значну гнучкість у виборі способу застосування засобів *IPSec* до трафіка *IP*. Захищені зв'язки можуть комбінуватися, щоб отримати бажану конфігурацію. Крім того, протокол *IPSec* забезпечує достатню вибірковість, розрізняючи трафік, що підлягає захисту, і трафік, якому дозволяється обійти його механізми за рахунок асоціації частини потоку даних *IP* з наявними захищеними зв'язками.

Засобом, за допомогою якого реалізується асоціація потоку *IP* із захищеними зв'язками, є номінальна база даних політики безпеки. У найбільш простій своїй формі база даних політики безпеки – це набір записів, кожен з яких визначає деяку підмножину потоку *IP* і вказує захищений зв'язок для цієї підмножини. У більш складних середовищах може визначатися декілька записів, які потенційно відповідають одному захищеному зв'язку, або множині захищених зв'язків, асоційованих з одним елементом бази даних політики безпеки. Кожен такий запис складається з набору селекторів (значень полів протоколу *IP* і протоколів вищого рівня). Селектори використовуються для фільтрації вихідного потоку з метою його відображення в конкретний захищений зв'язок. Кожен пакет, що відправляється, обробляється таким чином.

1. Порівнюються значення відповідних полів пакета (селекторів) з полями бази даних політики безпеки, знаходиться потрібний запис, у якому зазначено деяке (можливо, нульове) число захищених зв'язків.
2. Визначається захищений зв'язок і відповідний індекс параметрів захисту для даного пакета, якщо захист потрібний.
3. Виконуються необхідні операції *IPSec* (тобто обробка *AH* або *ESP*). Запис бази даних політики безпеки складається з таких селекторів.
   - **Адреса IP пункту призначення.** Це може бути одна адреса *IP*, список, діапазон адрес або група адрес (задається маскою). Останні два варіанти призначені для визначення декількох систем-адресатів, які використовують один захищений зв'язок.
   - **Ідентифікатор користувача** (*UserID*). Ідентифікатор користувача можна отримати від операційної системи. Це не поле в заголовку *IP* або заголовку протоколу верхнього рівня, проте воно доступно, коли *IPSec* виконується в тій же операційній системі, з якою працює користувач.
   - **Рівень (гриф) таємності даних.** Передбачений для систем, що забезпечують відповідний захист інформаційного потоку (наприклад, секретний або несекретний).

- **Протокол транспортного рівня**. Відповідна інформація береться з протоколу *IPv4* або поля *Next Header* (такий заголовок) протоколу *IPv6*. Це може бути конкретний номер протоколу, список номерів протоколів або діапазон номерів протоколів.
- **Протокол** *IPSec* (*AH, ESP* або *AH/ESP*). Якщо є, то береться з протоколу *IPv4* або з поля *Next Header* протоколу *IPv6*.
- **Порти джерела й адресата**. Це або конкретні значення портів *TCP* або *UDP*, нумерований список портів, або порт, поданий груповим символом.
- **Клас** *IPv6*. Відповідна інформація береться із заголовка *IPv6*. Це може бути конкретне значення для класу *IPv6* або значення, що задається груповим символом.
- **Мітка потоку** *IPv6*. Відповідна інформація береться із заголовка *IPv6*. Це може бути конкретне значення для мітки потоку *IPv6* або значення, що задається груповим символом.
- **Тип сервісу** *IPv4* (*Type of Service – TOS*). Відповідна інформація береться із заголовка *IPv4*. Це може бути конкретне значення для типу сервісу *IPv4* або значення, що задається груповим символом.

3.3 Транспортний і тунельний режими протоколу *IPSec*

Заголовки *AH* і *ESP* підтримують два режими використання: транспортний і тунельний.

Транспортний режим забезпечує захист насамперед для протоколів верхнього рівня. Це значить, що захист транспортного режиму поширюється на корисний вантаж пакета *IP*. Прикладами можуть бути сегменти *TCP*, *UDP* або *ICMP*, які розміщаються в стеці протоколів хоста безпосередньо над *IP*. Звичайно транспортний режим забезпечує наскрізний зв'язок двох вузлів. Коли система використовує заголовки *AH* або *ESP* над *IPv4*, корисним вантажем є дані, звичайно розташовувані відразу після заголовка *IP*, для *IPv6* – після заголовка *IP* і всіх наявних заголовків розширень *IPv6*, за винятком, можливо, заголовка параметрів адресата, який теж може підлягати захистові.

ESP у транспортному режимі шифрує і, якщо потрібно, автентифікує корисний вантаж *IP*, але не заголовок *IP*. *AH* у транспортному режимі припускає автентифікацію корисного вантажу і деяких частин заголовка *IP*.

Тунельний режим забезпечує захист усього пакета *IP*. Для виконання цієї задачі, після додавання до пакета *IP* полів *AH* або *ESP* весь пакет разом з полями захисту розглядається як корисний вантаж деякого нового "зовнішнього" пакета *IP* з новим зовнішнім заголовком *IP*. Весь оригінальний (внутрішній) пакет при цьому пересилається через "тунель" від однієї точки мережі *IP* до іншої, і жоден з маршрутизаторів на шляху не може перевірити внутрішній заголовок *IP*.

Оскільки оригінальний пакет інкапсульований в новий, більший

пакет, цей новий пакет може мати зовсім інші параметри джерела й адресата, що підсилює захист.

Тунельний режим використовується тоді, коли один або обидва кінці захищеного зв'язку є шлюзами захисту, наприклад, міжмережевими екранами або маршрутизаторами, що використовують *IPSec*. В цьому випадку розміщені за екранами мережі можуть здійснювати захищений обмін даними без застосування *IPSec*. Незахищені пакети, що генеруються такими системами, передаються по тунелях, прокладених через зовнішні мережі за допомогою захищених зв'язків у тунельному режимі, які встановлюються програмним забезпеченням *IPSec* міжмережевого екрана або захищеного маршрутизатора на межі локальної мережі.

Розглянемо приклад використання тунельного режиму *IPSec*. Вузол *A* мережі генерує пакет *IP* з адресою вузла призначення *B* іншої мережі. Цей пакет направляється від вузла *A*, до екрана або захищеного маршрутизатора на границі мережі *A*. Міжмережевий екран фільтрує усі вихідні пакети. Якщо прямуючий від *A* к *B* пакет вимагає захисту, екран виконує функції *IPSec* та інкапсулює оригінальний пакет у зовнішній пакет *IP*. Адресою *IP* відправника цього зовнішнього пакета буде даний міжмережевий екран, а адресою одержувача може бути екран, що формує границю локальної мережі *B*. Тепер пакет направляється екрану вузла *B*, а проміжні маршрутизатори будуть бачити лише зовнішній заголовок *IP*. У міжмережевому екрані вузла *B* зовнішній заголовок *IP* відділяється, а внутрішній пакет доставляється вузлові *B*.

ESP у тунельному режимі шифрує і, якщо потрібно, автентифікує весь внутрішній пакет *IP*, включаючи внутрішній заголовок *IP*. *AH* у тунельному режимі автентифікує весь внутрішній пакет *IP* і деякі частини зовнішнього заголовка *IP*. У табл. 3.2 подані функціональні можливості транспортного і тунельного режимів.

Таблиця 3.2 – Функціональні можливості транспортного і тунельного режимів

	Транспортний режим захищеного зв'язку	Тунельний режим захищеного зв'язку
AH	Автентифікує корисний вантаж *IP*, а також деякі частини заголовка *IP* і заголовків розширень *IPv6*	Автентифікує весь внутрішній пакет *IP* (заголовок і корисний вантаж внутрішнього пакета *IP*), а також деякі частини зовнішнього заголовка *IP* і зовнішніх заголовків розширень *IPv6*
ESP	Шифрує корисний вантаж *IP* і всі заголовки розширень *IPv6*	Шифрує внутрішній пакет *IP*
ESP з автентифікацією	Шифрує корисний вантаж *IP* і всі заголовки розширень *IPv6*. Автентифікує корисний вантаж *IP*, але не заголовок *IP*	Шифрує внутрішній пакет *IP*. Автентифікує внутрішній пакет *IP*

3.4 Використання заголовку автентифікації

Заголовок автентифікації (*AH*) забезпечує підтримку цілісності даних і автентифікацію пакетів *IP*. Перше гарантує неможливість таємної модифікації вмісту пакета на шляху проходження. Функція автентифікації дозволяє кінцевій системі або мережевому пристрою ідентифікувати користувача або програмний додаток і, відповідно, відфільтрувати трафік та захиститися від дуже розповсюджених сьогодні в *Інтернет* атак з підміною мережевих адрес. Заголовок *AH* захищає також від атак відтворення повідомлень, мова про які піде нижче.

Автентифікація опирається на використання кодів автентичності повідомлень, при цьому дві сторони повинні використовувати загальний секретний ключ. Заголовок автентифікації складається з полів показаних на рис. 3.3.

Такий заголовок (8 біт). Ідентифікує тип заголовка, що знаходиться безпосередньо за даним заголовком.

Довжина корисного вантажу (8 біт). Довжина заголовка автентифікації в 32-бітових словах, зменшена на 2. Наприклад, установлена за замовчуванням довжина поля чи даних автентифікації дорівнює 96 біт, або трьом 32-бітовим словам. Разом із заголовком фіксованої довжини в три слова загальна довжина всього заголовка виявляється рівна 6 словам, тому в полі довжини корисного вантажу в цьому випадку вказується значення 4.

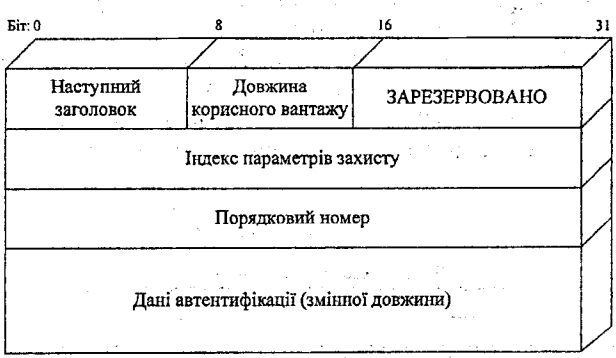


Рисунок 3.3 – Заголовок автентифікації IPSec

Зарезервовано (16 біт). Для майбутнього використання.

Індекс параметрів захисту (32 біт). Ідентифікує захищений зв'язок.

Порядковий номер (32 біт). Значення лічильника.

Дані автентифікації (змінної довжини). Поле змінної довжини (рівне цілому числу 32-бітових слів), яке містить код *ICV* (*Integrity Check Value* – код контролю цілісності) або код *MAC* (*Message Authentication Code* – код автентичності повідомлення) для даного пакета.

3.4.1 Сервіс захисту від відтворення

Атака відтворення повідомлень полягає у тому, що зловмисник може одержати екземпляр пакета і пізніше пред'явити його певному адресатові. Повторне одержання однакових пакетів *IP* може якимось чином порушити сервіс або мати інші небажані наслідки. Поле порядкового номера пропонується використовувати саме для того, щоб усунути можливість проведення такої атаки.

Коли встановлюється новий захищений зв'язок, відправник ініціалізує лічильник порядкових номерів, установивши відповідне значення рівним 0. Щоразу, коли по відповідному захищеному зв'язку посилається пакет, відправник збільшує значення даного лічильника і розміщує його в поле порядкового номера. Таким чином, першим значенням буде 1. Якщо активована функція захисту від відтворення (що передбачається за замовчуванням), відправник не повинен дозволити порядковому номерові перевершити значення $2^{32}-1$, після якого значення лічильника скидається в нуль, оскільки тоді принаймні два пакети будуть мати один і той самий порядковий номер. Якщо межа $2^{32}-1$ виявляється досягнутою, відправник повинен завершити даний захищений зв'язок і почати переговори про створення нового захищеного зв'язку з новим ключем.

Оскільки протокол *IP* припускає обмін без встановлення з'єднань, він не гарантує, що пакети будуть доставлені в необхідному порядку і без втрат. Тому автентифікація *IPSec* вимагає, щоб одержувач мав спеціальне вікно розміром *W* (за замовчуванням *W*=64). Права границя вікна повинна являти собою найвищий порядковий номер *N* для вже отриманих дійсних пакетів. Для будь-якого пакета з порядковим номером з діапазону від *(N-W+1)* до *N*, що прийнятий коректно (тобто належним чином автентифікований), позначається відповідний сегмент вікна (рис. 3.4).

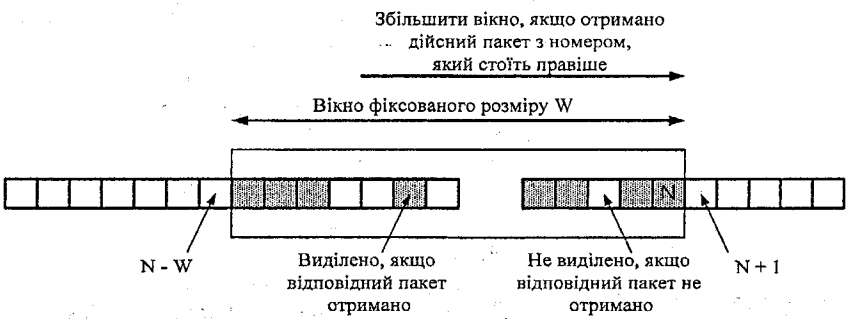


Рисунок 3.4 – Механізм захисту від відтворення

При одержанні пакета його обробка виконується в такий спосіб:

- якщо отриманий пакет попадає в рамки вікна і є новим, перевіряється значення *MAC*. Якщо автентифікація пакета завершується успішно, позначається відповідний сегмент у вікні;

- якщо отриманий пакет попадає в область правіше від вікна і є новим, перевіряється значення *MAC*. Якщо автентифікація пакета завершується успішно, вікно розширюється так, щоб новий порядковий номер виявився правою границею вікна, а у вікні позначається відповідний сегмент;
- якщо отриманий пакет виявляється ліворуч від вікна або автентифікувати його не вдається, він буде відкинутий. Така подія повинна реєструватися.

3.4.2 Код контролю цілісності

Поле даних автентифікації містить значення, назване кодом *ICV*. Код *ICV* являє собою код автентичності повідомлення або усічену версію такого коду, породженого алгоритмом *MAC*. Наявні на сьогодні специфікації протоколу потребують, щоб будь-яка реалізація підтримувала такі схеми *HMAC-MD5-96*, *HMAC-SHA-1-96*.

Обидві схеми припускають застосування алгоритму *HMAC*, у першому випадку з використанням хеш-коду *MD5*, а в другому – хеш-коду *SHA-1*. В обох випадках обчислюється повне значення *HMAC*, але потім воно усікається до перших 96 біт, що відповідає довжині поля даної автентифікації, установленої за замовчуванням.

Для обчислення значення *MAC* береться така інформація:

- поля заголовка *IP*, що або не змінюються при передачі (незмінні поля), або мають прогнозовані значення в пункті призначення захищеного зв'язку *AH*. Поля, що можуть змінитися при передачі, і значення яких у кінцевій точці не можна прогнозувати, обнуляються для обчислень і в пункті відправлення, і в пункті призначення;
- заголовок *AH*, відмінний від поля даних автентифікації. Поле даних автентифікації обнуляється для обчислень і в пункті відправлення, і в пункті призначення;
- усі дані такого протоколу верхнього рівня, що повинні залишатися незмінними при передачі (наприклад, сегмент *TCP* або внутрішній пакет *IP* у тунельному режимі).

Для *IPv4* прикладами незмінних полів є поля довжини заголовка *Інтернет* та адреси джерела. Прикладом змінюваного, але прогнозованого поля, є поле адреси одержувача (з наближеним або строгим трасуванням від джерела повідомлення). Прикладами змінюваних полів, що обнуляються перед обчисленням *ICV*, є поля часу існування і контрольної суми заголовка. Помітимо, що поля адреси як джерела, так і адресата захищаються, так що їхня фальсифікація виключена.

Для *IPv6* прикладами відповідних типів полів у базовому заголовку є поле версії (незмінна), адреса одержувача (змінювана, але прогнозована) і мітка потоку (змінювана та обнулена для обчислень).

3.4.3 Транспортний і тунельний режими при автентифікації

На рис. 3.5 показано два варіанти використання сервісу автентифікації *IPSec*.

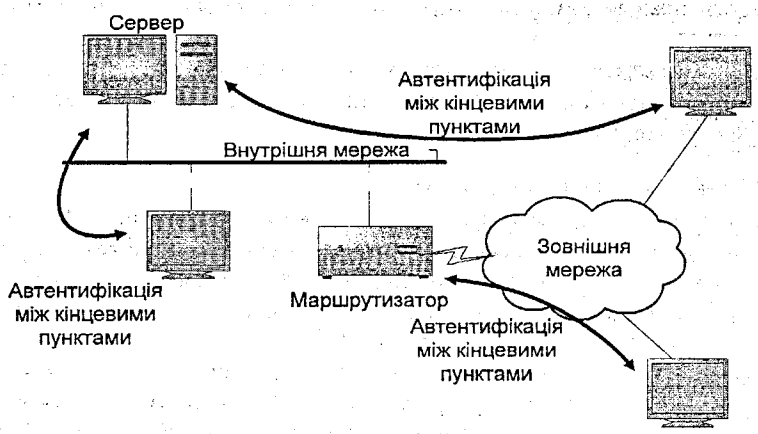


Рисунок 3.5 – Автентифікація між кінцевими пунктами і автентифікація між кінцевим і проміжним пунктами

У першому випадку автентифікація виконується безпосередньо між сервером і робочими станціями клієнтів, причому робочі станції можуть розміщатися як у тій же мережі, що і сервер, так і в зовнішній мережі.

Для транспортного режиму *AH* із застосуванням *IPv4* дані *AH* розміщаються безпосередньо за оригінальним заголовком *IP* і перед корисним вантажем *IP*, як показано у верхній частині рисунку 3.6, б. Автентифікації підлягає весь пакет, за винятком змінюваних полів у заголовку *IPv4*, які при обчисленні значення *MAC* обнулюються.

У контексті *IPv6* дані *AH* розглядаються як корисний вантаж наскрізної передачі, тобто перевірка й обробка цих даних проміжними маршрутизаторами не передбачається. Тому дані *AH* розміщаються після базового заголовка *IPv6* і заголовків розширень транзиту, маршрутизації і фрагментації. Заголовок розширення параметрів адресації може розміщатися до і після заголовка *AH* – в залежності від вимог семантики. Як і для *IPv4* автентифікація передбачається для всього пакета, за винятком змінюваних полів, що при обчисленні значення *MAC* обнулюються.

Для тунельного режиму *AH* автентифікації підлягає весь оригінальний пакет *IP*, а заголовок *AH* розміщається між оригінальним заголовком *IP* і новим зовнішнім заголовком *IP* (рис. 3.6, в). Внутрішній заголовок *IP* містить адреси оригінальних джерела й адресата, у той час як зовнішній заголовок може містити зовсім інші адреси *IP* (наприклад, адреси міжмережевого екрана або інших шлюзів захисту).

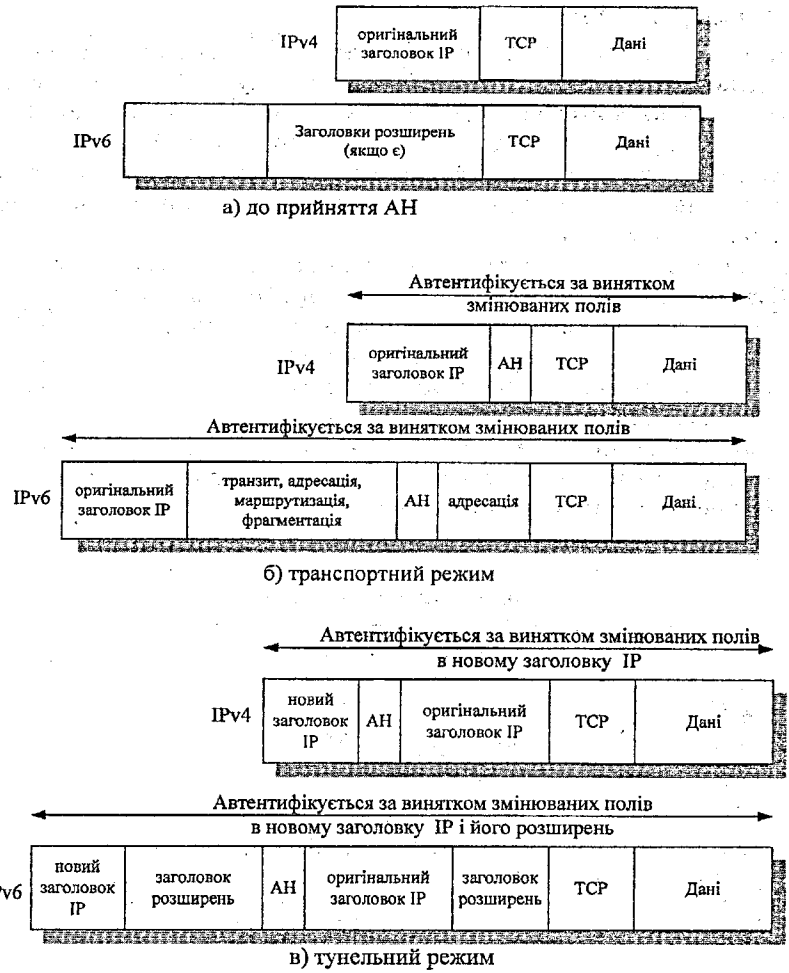


Рисунок 3.6 – Область застосування автентифікації АН

У тунельному режимі весь внутрішній пакет *IP*, включаючи внутрішній заголовок, захищається засобами *АН*. Зовнішній заголовок *IP* захищається винятком змінюваних і не прогнозованих за значенням полів (для *IPv6* це стосується і зовнішніх заголовків розширень *IP*).

3.5 Захищений корисний вантаж

Протокол *ESP* забезпечує сервіс конфіденційності, у тому числі захист змісту повідомлення й обмежену конфіденційність транспортного потоку. Додатково *ESP* може забезпечувати сервіс автентифікації як *АН*.

3.5.1 Формат *ESP*

На рисунку 3.7 показаний формат пакета *ESP*. Він містить такі поля.

Індекс параметрів захисту (32 біт). Ідентифікує захищений зв'язок.

Порядковий номер (32 біт). Значення лічильника, який використовується для захисту від атак відтворення, (як і в протоколі *AH*).

Корисний вантаж (змінної довжини). Це сегмент транспортного рівня (у транспортному режимі) або пакет *IP* (у тунельному режимі), який захищається шифруванням.

Заповнювач (0–255 байт). Призначення поля описується нижче.

Довжина заповнювача (8 біт). Вказує число байтів заповнювача.

Наступний заголовок (8 біт). Ідентифікує тип даних, які містяться в полі даних корисного вантажу, указуючи перший заголовок корисного вантажу (заголовок розширення *IPv6* або заголовок протоколу верхнього рівня, наприклад *TCP*).

Дані автентифікації (змінної довжини). Поле змінної довжини (ціле число 32-бітових слів), містить код *ICV*, який обчислюється для всього пакета *ESP* без поля даних автентифікації.

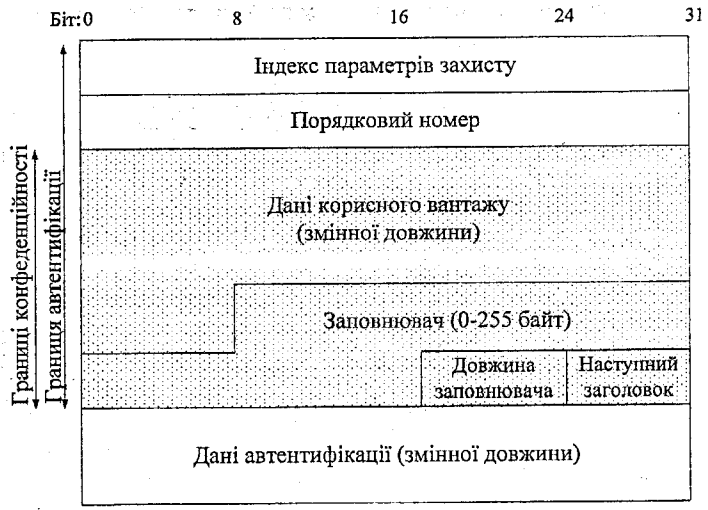


Рисунок 3.7 – Формат ESP для IPSec

3.5.2 Шифрування й алгоритми автентифікації

Сервіс *ESP* припускає шифрування полів корисного вантажу, заповнювача, довжини заповнювача і заголовка. Якщо для алгоритму, використовуваного при шифруванні корисного вантажу, потрібна криптографічна синхронізація даних (наприклад, вектор ініціалізації *IV*), то необхідні дані можуть бути перенесені в початок поля або корисного

вантажу. Значення вектора ініціалізації не шифрується, хоча часто вважається частиною шифрованого тексту.

Існуючі специфікації вимагають, щоб будь-яка реалізація підтримувала використання алгоритму *DES* у режимі *CBC* (режим зчеплення шифрованих блоків). У документації визначається ряд ідентифікаторів для інших алгоритмів, які також можуть застосуються для шифрування ("потрійний" *DES* із трьома ключами, *RC5*, *IDEA*, "потрійний" *IDEA* із трьома ключами, *CAST*, *Blowfish*).

Як і протокол *AH*, протокол *ESP* підтримує використання значень *MAC*, довжина яких за замовчуванням дорівнює 96 біт та схеми *HMAC-MD5-96* і *HMAC-SHA-1-96*.

3.5.3 Використання заповнювача

Якщо алгоритм шифрування потребує, щоб довжина відкритого тексту була кратна деякому цілому числу байтів (наприклад, довжині одного блока блокового шифру), поле заповнювача використовується для того, щоб доповнити до потрібної довжини відкритий текст, який складається з полів корисного вантажу, заповнювача, довжини заповнювача і заголовка.

Формат *ESP* потребує, щоб поле довжини заповнювача і заголовка вирівнювалися по правому краю в 32-бітовому слові. Це еквівалентно вимозі, щоб шифрований текст мав довжину, кратну 32 бітам. Поле заповнювача призначене для здійснення такого вирівнювання.

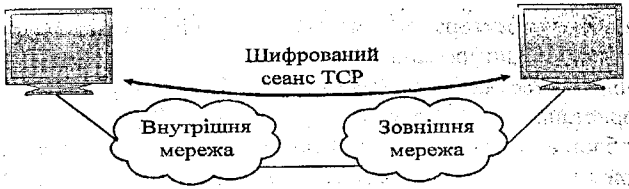
Заповнення можна використовувати і для того, щоб забезпечити часткову конфіденційність транспортного потоку шляхом маскування інформації про справжню довжину корисного вантажу.

3.5.4 Транспортний і тунельний режими в сервісі *ESP*

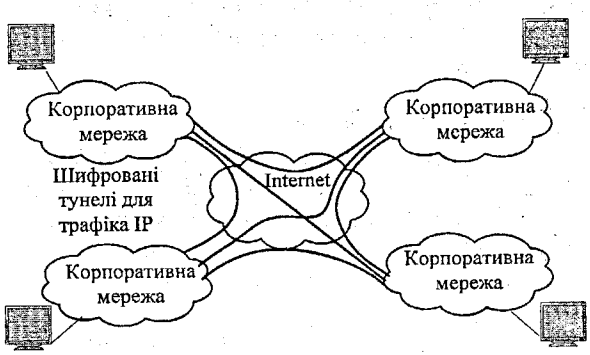
На рисунку 3.8 показано два варіанти застосування сервісу *ESP* протоколу *IPSec*. У верхній частині рисунка шифрування (і, як опція, автентифікація) здійснюється безпосередньо між двома вузлами (рис 3.8, а).

На рисунку 3.8, *б* показано, як використовувати тунельний режим для побудови віртуальної приватної мережі. У цьому прикладі передбачається, що деяка організація має чотири ділянки мережі, пов'язані через *Інтернет*. Вузли внутрішніх мереж використовують *Інтернет* для передачі даних, але не взаємодіють з іншими розміщеними в *Інтернет* вузлами. Оскільки тунелі закінчуються в шлюзах захисту кожної внутрішньої мережі, така конфігурація дозволяє внутрішнім вузлам цих мереж уникнути необхідності застосування механізмів захисту. У першому варіанті застосовується транспортний режим захищеного зв'язку, а в другому – тунельний режим.

Як і при використанні протоколу *AH*, для опису області дії *ESP* застосовуються пакети у форматі, подібному до показаного на рисунку 3.6, а.



а) шифрування між двома вузлами



б) шифрування у віртуальній приватній мережі

Рисунок 3.8 – Шифрування в транспортному і тунельному режимах

3.5.5 Транспортний режим протоколу *ESP*

Транспортний режим *ESP* використовується для шифрування і, якщо потрібно, автентифікації даних, які пересилаються в пакеті *IP* (наприклад, сегмента *TCP*, як показано на рисунку 3.9, *а*). Для цього режиму в рамках *IPv4* заголовок *ESP* розміщається в пакеті *IP* безпосередньо перед заголовком транспортного рівня (наприклад, *TCP, UDP, ICMP*), а кінець пакета *ESP* (заповнювач, довжина заповнювача і заголовка) – після пакета *IP*.

Якщо використовується функція автентифікації, то поле даних автентифікації додається в кінець *ESP*. Весь сегмент транспортного рівня разом з кінцем *ESP* шифруються. Автентифікація охоплює весь шифрований текст і заголовок *ESP*.

У контексті *IPv6* дані *ESP* розглядаються як призначений для наскрізного пересилання корисний вантаж, який не підлягає перевірці або обробці проміжними маршрутизаторами. Тому заголовок *ESP* розташовується після основного заголовка *IPv6* і заголовків розширень транзиту, маршрутизації та фрагментації. Заголовок розширення параметрів адресата може бути розташований до або після заголовка *ESP* – в залежності від вимог семантики.

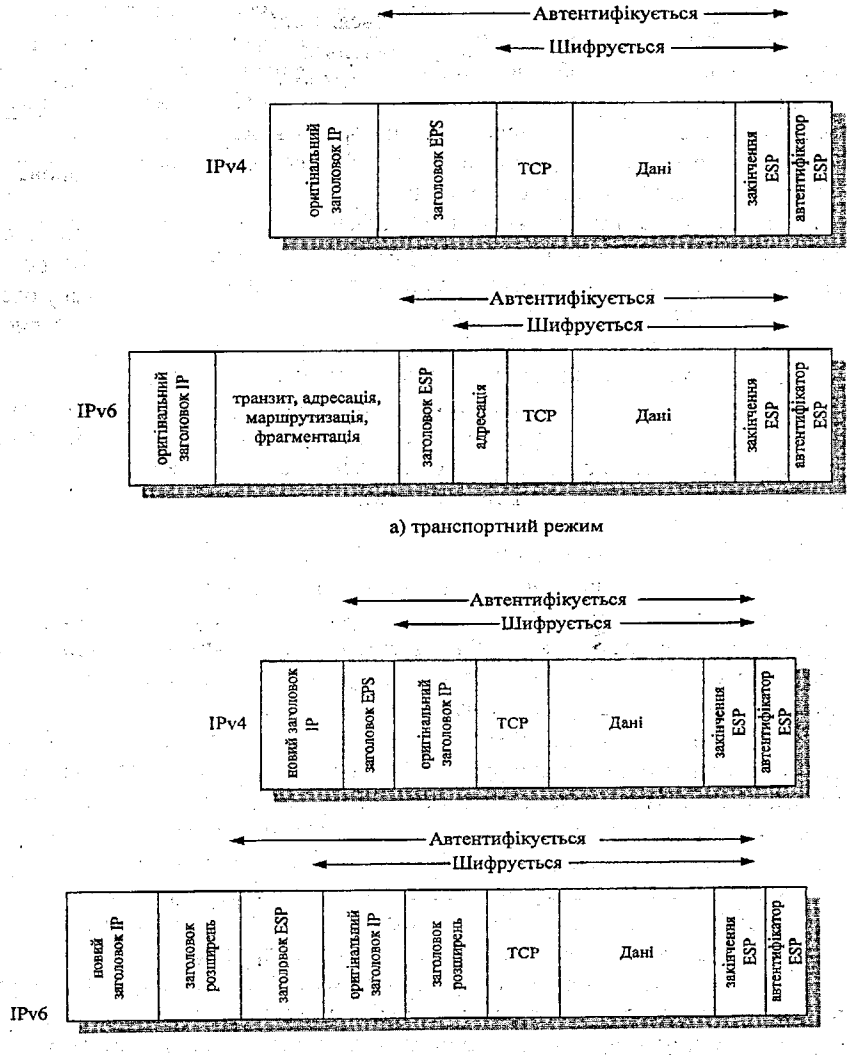


Рисунок 3.9 – Область дії шифрування та автентифікації ESP

У *IPv6* шифрування охоплює весь сегмент транспортного рівня разом з кінцевим *ESP*, а також заголовок розширення параметрів адресата, якщо цей заголовок розміщається після заголовка *ESP*. Автентифікація передбачається для шифрованого тексту і заголовка *ESP*.

У вузлі джерела блок даних, який складається з кінця *ESP* і всього сегмента транспортного рівня, шифрується, і відкритий текст цього блока замінюється шифрованим текстом, у результаті формується пакет *IP* для пересилання. Якщо обрано опцію автентифікації, то додається поле автентифікації. Пакет направляється адресатові. Кожен проміжний маршрутизатор повинний перевірити й обробити заголовок *IP*, а також усі заголовки розширень *IP*, доступні в нешифрованому вигляді. Шифрований текст при цьому залишається незмінним.

Вузол адресата перевіряє й обробляє заголовок *IP* і всі заголовки розширень *IP*, доступні в нешифрованому вигляді. Потім на основі інформації індексу параметрів захисту в заголовку *ESP* дешифруються інші частини пакета, у результаті чого стає доступним сегмент транспортного рівня у вигляді відкритого тексту. Транспортний режим забезпечує конфіденційність та дозволяє уникнути необхідності реалізації функцій захисту в кожному окремому програмному додатку. Цей режим досить ефективний, а обсяг пакета збільшується незначно. Недоліком цього режиму є те, що при його використанні не виключається можливість аналізу трафіка пакетів, що пересилаються.

3.5.6 Тунельний режим *ESP*

Тунельний режим *ESP* припускає шифрування всього пакета *IP* (рисунок 3.9, *б*). У цьому режимі заголовок *ESP* додається до пакета як префікс, а потім пакет разом з кінцевим *ESP* шифруються. Даний метод можна використовувати, коли потрібно усунути можливість проведення атак, побудованих на аналізі трафіка.

Оскільки заголовок *IP* містить адресу пункту призначення і, можливо, директиви вихідної маршрутизації разом з інформацією про параметри транзиту, не можна просто передати шифрований пакет *IP* з доданим до нього у вигляді префікса заголовком *ESP*. Проміжні маршрутизатори не зможуть обробити такий пакет. Таким чином, необхідно включити весь блок (заголовок *ESP*, шифрований текст і дані автентифікації, якщо вони є, у зовнішній пакет *IP* з новим заголовком, який буде містити достатньо інформації для маршрутизації, але не для аналізу трафіка.

У той час як транспортний режим підходить для захисту з'єднань між вузлами, які підтримують сервіс *ESP*, тунельний режим виявляється корисним у конфігурації, що припускає наявність міжмережевого екрана або іншого шлюзу захисту внутрішньої мережі від зовнішніх.

У тунельному режимі шифрування використовується для обміну тільки між зовнішнім вузлом і шлюзом захисту або між двома шлюзами захисту. Це розвантажує вузли внутрішньої мережі (немає необхідності шифрування даних) і спрощує процедуру розподілу ключів, зменшуючи їх число. Крім того, такий підхід ускладнює аналіз потоку повідомлень, який направляється конкретному адресатові.

Розглянемо випадок, коли зовнішній вузол з'єднується з вузлом внутрішньої мережі, захищеної міжмережевим екраном, і сервіс *ESP* використовується зовнішнім вузлом і екраном. Тоді при пересиланні сегмента транспортного рівня від зовнішнього вузла до вузла внутрішньої мережі виконуються такі дії.

Джерело готує внутрішній пакет *IP* із зазначенням адреси пункту призначення (вузла внутрішньої мережі). До цього пакета у вигляді префікса додається заголовок *ESP*. Потім пакет і кінець *ESP* шифруються і до результату можуть бути додані дані автентифікації. Отриманий блок ховається в зовнішній пакет *IP* з новим заголовком *IP* (базовий заголовок плюс необов'язкові розширення, наприклад, параметрів маршрутизації і транзиту для *IPv6*), у якому адресою пункту призначення є адреса міжмережевого екрана.

Зовнішній пакет відправляється міжмережевому екрану. Кожен проміжний маршрутизатор повинен перевірити та обробити зовнішній заголовок *IP* і всі зовнішні заголовки розширень *IP*, залишаючи при цьому шифрований текст незмінним.

Екран-адресат перевіряє та обробляє зовнішній заголовок *IP* і всі зовнішні заголовки розширень *IP*. Потім на основі інформації, наданої індексом параметрів захисту в заголовку *ESP*, міжмережевий екран дешифрує інші частини пакета, у результаті чого стає доступним внутрішній пакет *IP* у вигляді відкритого тексту. Цей пакет потім передається по внутрішній мережі. Внутрішній пакет передається маршрутизаторові внутрішньої мережі або безпосередньо вузлові-адресатові.

3.6 Комбінації захищених зв'язків

Певний захищений зв'язок може використовувати або протокол *AH*, або *ESP*, але в жодному разі не обидва одночасно. Проте, іноді конкретному потокові даних може бути необхідним і сервіс *AH*, і сервіс *ESP*. Крім того, конкретному потокові даних може знадобитися сервіс *IPSec* для зв'язку між вузлами та інший сервіс для зв'язку між шлюзами захисту, наприклад міжмережевими екранами. В усіх цих випадках одному потокові для одержання всього комплексу послуг *IPSec* потрібно кілька захищених зв'язків. Тут корисним виявляється поняття «пучок захищених зв'язків» (*security association bundle*), що визначає набір захищених зв'язків, за допомогою яких потокові повинен надаватися необхідний набір послуг *IPSec*. При цьому захищені зв'язки в пучку можуть завершуватися в різних кінцевих точках. Захищені зв'язки можуть бути об'єднані в пучки двома способами.

Транспортна суміжність. Застосування кількох протоколів захисту до одного пакета *IP* без створення тунелю. Цей підхід до створення комбінації *AH* і *ESP* виявляється ефективним тільки для одного рівня вкладення: подальші вкладення не дають додаткового виграшу, оскільки обробка

виконується в одній інстанції – протоколі *IPSec*(кінцевого) одержувача.

Повторне тунелювання. Застосування декількох рівнів протоколів захисту за допомогою тунелювання *IP*. Цей підхід допускає множину рівнів вкладення, оскільки тунелі можуть починатися і завершуватися в різних вузлах мережі, які використовують *IPSec,* по маршруту передачі даних.

Ці два підходи можна й об'єднати (наприклад, організувавши в частині тунельного захищеного зв'язку між шлюзами захисту, транспортний захищений зв'язок між вузлами).

Цікавим питанням, що виникає при розгляді пучків захищених зв'язків, є питання про порядок, у якому можуть застосовуватися автентифікація і шифрування між даною парою кінцевих вузлів, і способи здійснення цих операцій.

Шифрування й автентифікація можуть комбінуватися для того, щоб забезпечити переданому пакетові *IP* як конфіденційність, так і автентифікацію. Ми розглянемо кілька підходів до такого комбінування.

3.6.1 *ESP* з опцією автентифікації

Користувач спочатку застосовує сервіс *ESP* до даних, які потребують захисту, а потім додає поле даних автентифікації. В цьому випадку можна виділити такі дві можливості.

Транспортний режим *ESP*. Автентифікація і шифрування застосовуються до корисного вантажу *IP*, що доставляється вузлові адресата, але при цьому заголовок IP не захищається.

Тунельний режим *ESP*. Автентифікація застосовується до всього пакета *IP*, що доставляється за адресою *IP* зовнішнього одержувача (наприклад міжмережевого екрана), і автентифікація виконується цим одержувачем. Весь внутрішній пакет *IP* захищається механізмом таємності, оскільки призначено для доставки внутрішньому адресатові *IP*.

В обох випадках автентифікації підлягає шифрований текст.

3.6.2 Транспортна суміжність

Іншим способом застосування автентифікації після шифрування є використання пучка з двох захищених зв'язків у транспортному режимі, де внутрішній зв'язок є захищеним зв'язком *ESP*, а зовнішній – захищеним зв'язком *AH*. У цьому випадку *ESP* використовується без опції автентифікації. Оскільки внутрішній захищений зв'язок використовується в транспортному режимі, шифруванню підлягає корисний вантаж *IP*. Отриманий у результаті пакет складається із заголовка *IP* (і, можливо, заголовків розширень *IPv6*), за яким слідують дані *ESP*. Потім застосовується сервіс *AH* у транспортному режимі, так що автентифікація буде охоплювати дані *ESP* і оригінальний заголовок *IP* (а також розширення), за винятком змінюваних полів. Перевага цього підходу в порівнянні з простим використанням захищеного зв'язку *ESP* з опцією автентифікації полягає в

тому, що при комбінованому підході автентифікація охоплює більше полів, включаючи поля адрес *IP* джерела й адресата. Недолік пов'язаний з використанням двох захищених зв'язків замість одного.

3.6.3 Транспортно-тунельний пучок

Використання функцій автентифікації до шифрування може виявитися більш кращим з кількох причин. По-перше, оскільки дані автентифікації будуть захищені шифруванням, ніхто сторонній не зможе перехопити повідомлення та таємно змінити дані автентифікації. По-друге, якщо необхідно зберігати інформацію автентифікації разом з повідомленням у системі адресата, то зручніше автентифікувати відкрите повідомлення. Інакше для повторної автентифікації повідомлення прийдеться повторно шифрувати.

Варіантом застосування автентифікації до шифрування при обміні даними між двома вузлами є використання пучка захищених зв'язків, який складається з внутрішнього захищеного транспортного зв'язку *AH* та зовнішнього захищеного тунельного зв'язку *ESP*. У цьому випадку автентифікація охоплює корисний вантаж *IP* разом із заголовком *IP* (та розширеннями), за винятком змінюваних полів. Отриманий у такий спосіб пакет *IP* потім обробляється в тунельному режимі *ESP*, у результаті чого внутрішній пакет разом з даними автентифікації стає зашифрованим і має новий заголовок *IP* (з необхідними розширеннями).

3.6.4 Основні типи комбінацій захищених зв'язків

У документі, що описує архітектуру *IPSec*, наведено чотири приклади комбінації захищених зв'язків, що повинні підтримуватися вузлами, які використовують *IPSec* (робочими станціями, серверами) або шлюзами захисту (міжмережевими екранами, маршрутизаторами). Ці приклади ілюструються на рисунку 3.10. У нижній частині рисунка для кожного випадку зображені схеми фізичного з'єднання елементів, а у верхній частині – схеми логічного зв'язування за допомогою одного або декількох вкладених захищених зв'язків. Кожен захищений зв'язок може бути або зв'язком *AH*, або *ESP*. Для захищених зв'язків між вузлами використовується транспортний або тунельний режим, в інших випадках – тільки тунельний.

Варіант 1 забезпечує захист зв'язку між системами, які використовують *IPSec*. Дві системи, які обмінюються повідомленнями через захищений зв'язок, повинні використовувати загальні секретні ключі. При цьому допустимі такі комбінації.

а) *AH* у транспортному режимі;

б) *ESP* у транспортному режимі;

в) спочатку *AH*, а потім *ESP* у транспортному режимі (захищений зв'язок *AH*, вкладений в захищений зв'язок *ESP*);

г) кожен зі зв'язків, зазначених у пунктах *а, б* та *в*, усередині *AH* або

ESP у тунельному режимі.

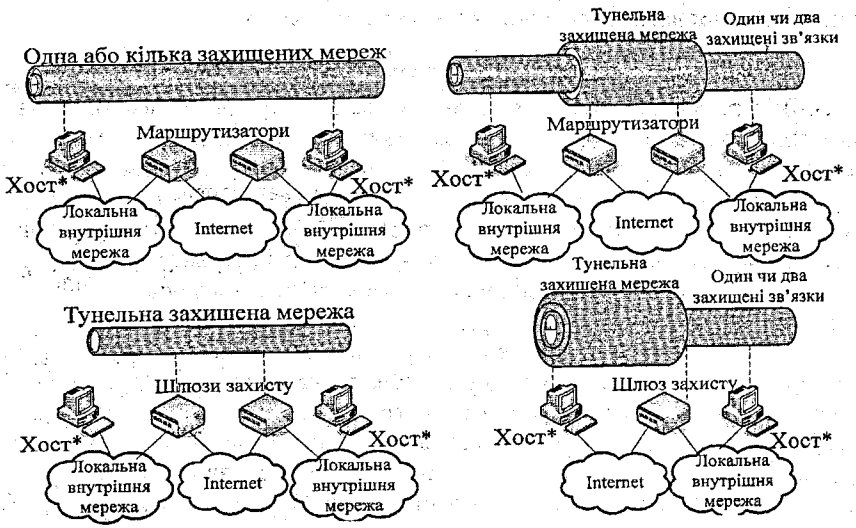


Рисунок 3.10 – Основні типи комбінацій захищених зв'язків

Варіант 2 забезпечує захист тільки між шлюзами (маршрутизаторами, екранами тощо), а в кінцевих вузлах застосування *IPSec* не передбачається. Ця ситуація ілюструє підтримку простої віртуальної приватної мережі. У документі, що описує архітектуру захисту, говориться, що в подібній ситуації потрібно тільки один тунельний захищений зв'язок. Тунель може використовувати *AH*, *ESP* або *ESP* з опцією автентифікації. Вкладені тунелі не потрібні, оскільки сервіс *IPSec* звертається до усього внутрішнього пакета.

Варіант 3 використовує схему варіанта 2 з додаванням наскрізного захисту. При цьому припустимими є ті ж комбінації, що вже були описані вище для варіантів 1 і 2. Тунель від шлюзу до шлюзу забезпечує автентифікацію або конфіденційність для усього графіка між кінцевими системами. Коли тунель від шлюзу до шлюзу використовує *ESP*, забезпечується й обмежена конфіденційність потоку обміну даними. Кінцеві системи можуть при цьому звертатися до додаткових сервісів *IPSec*, необхідних для конкретних додатків або окремих користувачів, використовуючи для цього наскрізні захищені зв'язки.

Варіант 4 ілюструє підтримку віддаленого вузла, що використовує *Інтернет* для зв'язку з міжмережевим екраном організації з метою одержання доступу до сервера або робочої станції за цим екраном. Між віддаленим вузлом і екраном необхідно використовувати тільки тунельний режим. Як і при варіанті 1, між віддаленим і локальним вузлами можна

використовувати один або два захищені зв'язки.

3.7 Керування ключами в протоколі *IPSec*

Функції керування ключами *IPSec* припускають визначення і розподіл секретних ключів. Для захищеного зв'язку між двома вузлами потрібно чотири ключі: по парі ключів передавання та одержання даних для сервісів *AH* і *ESP*. У документі, що описує архітектуру *IPSec*, говориться про підтримку таких двох типів керування ключами.

Ручне керування. Системний адміністратор вручну конфігурує кожну систему, вказуючи її ключі та ключі інших систем. Це цілком прийнятно для невеликих та статичних середовищ.

Автоматизоване керування. Автоматизована система забезпечує можливість створення ключів для захищених зв'язків за запитом та спрощує використання ключів у великій розподіленій системі з часто змінною конфігурацією.

Застосовуваний за замовчуванням для *IPSec* протокол автоматизованого керування ключами називається *ISAKMP/Oakley*.

3.7.1 Протокол *Oakley*

Oakley являє собою варіант обміну ключами, який виконується за удосконаленою схемою Діффі-Хелмана. Нагадаємо, що алгоритм Діффі-Хелмана пропонує таку взаємодію між користувачами *A* та *B*. Передбачається попереднє узгодження двох глобальних параметрів: q, досить велике просте число, і α, первісний корінь q. Сторона *A* вибирає випадкове ціле число X_A, яке буде особистим ключем *A*, і передає стороні *B* відкритий ключ $Y_A = \alpha^{X_A}$. Так само сторона *B* вибирає випадкове ціле число X_B, що буде особистим ключем *B*, і передає стороні *A* відкритий ключ $Y_B = \alpha^{X_B}$. Кожна зі сторін може тепер обчислити секретний сеансовий ключ.

Алгоритм Діффі-Хелмана має дві властивості.

1. Секретні ключі створюються тільки тоді, коли це потрібно. Немає необхідності зберігати секретні ключі протягом тривалого періоду часу, роблячи їх тим самим більш вразливими.

2. Процедура обміну не вимагає ніякої спеціальної підготовки, крім угоди про значення глобальних параметрів.

Однак, алгоритм Діффі-Хелмана не позбавлений і недоліків.

1. Алгоритм не пропонує методів, які б дозволили ідентифікувати сторони.

2. Алгоритм вразливий до атак з використанням посередника, коли деяка третя сторона *C* виступає від імені *B* для *A* і від імені *A* для *B*.

3. Алгоритм вимагає інтенсивних обчислень. В результаті він виявляється вразливим до атак засмічення, коли зловмисник запитує дуже велике число ключів. У підсумку жертва витрачає значну частину обчислювальних ресурсів на марні піднесення до степеня в арифметиці класів відрахувань.

Oakley розроблений саме для того, щоб, зберігши переваги алгоритму Діффі-Хелмана, позбутися від виявлених в останньому недоліків.

3.7.2 Властивості *Oakley*

Алгоритм *Oakley* характеризується такими п'ятьма важливими особливостями.

1. Використовує механізм так званих *cookies*, щоб протистояти атакам засмічення.

2. Дозволяє двом сторонам домовиться про групу, яка визначає глобальні параметри алгоритму обміну ключами Діффі-Хелмана.

3. Використовує оказії, щоб протистояти атакам відтворення повідомлень.

4. Здійснює обмін значеннями відкритих ключів Діффі-Хелмана.

5. Виконує автентифікацію обміну Діффі-Хелмана, щоб протистояти атакам посередника.

При атаці засмічення зловмисник фальсифікує адресу законного джерела і посилає жертві відкритий ключ Діффі-Хелмана. Жертва виконує модульне піднесення до степеня в арифметиці класів відрахувань, щоб обчислити секретний ключ. Багаторазово повторені повідомлення такого типу можуть засмітити систему сторони, що атакується, марною роботою. Вимога обміну *cookies* означає, що кожна зі сторін повинна послати в початковому повідомленні псевдовипадкове число, *cookie*, яке інша сторона повинна підтвердити. Це підтвердження повинне повторитися в першому повідомленні обміну ключами за алгоритмом Діффі-Хелмана. Якщо адреса джерела була фальсифікована, зловмисник відповіді не отримає. Таким чином, зловмисник зможе змусити користувача генерувати підтвердження, а не виконувати обчислення з використанням алгоритму Діффі-Хелмана.

ISAKMP вимагає, щоб процедура генерування *cookies* задовольняла такі три основні вимоги.

1. *Cookie* повинний залежати від параметрів сторони, що його генерує. Це не дасть можливості зловмисникові одержати *cookie* за допомогою реальної адреси *IP* і порту *UDP*, а потім використовувати цей *cookie* для того, щоб закидати жертву запитами з обраних випадковим чином *IP*-адрес або портів.

2. Об'єкт, що генерує, повинен використовувати при створенні і такій верифікації *cookies* локальну секретну інформацію. Цю інформацію неможливо отримати з будь-якого іншого *cookie*. Зміст цієї вимоги полягає в тому, щоб об'єктові, що генерує, не приходилося зберігати копії *cookies*, які у такий спосіб стають більш вразливими, і можна було перевірити *cookie*, що надходить з підтвердженням, коли це буде потрібно.

3. Генерування *cookies* і їхня верифікація повинні виконуватися швидко, щоб не допустити проведення атак, спрямованих на блокування

ресурсів системи.

Метод створення *cookies* полягає у швидкому обчисленні хеш-коду (наприклад *MD5*) для адрес *IP* джерела й адресата, портів *UDP* джерела й адресата і локально генерованого секретного значення.

Алгоритм *Oakley* підтримує використання різних груп для обміну ключами Діффі-Хеллмана. У кожній групі визначаються два глобальних параметри та алгоритм. Наявні специфікації визначають такі групи.

- Піднесення до степеня в арифметиці класів відрахувань з 768-бітовим модулем:
  $q = 2^{768} - 2^{704} - 1 + 2^{64} \times \left(\lfloor 2^{638} \times \pi \rfloor + 149686 \right)$,
  $\alpha = 2$.
- Піднесення до степеня в арифметиці класів відрахувань з 1024-бітовим модулем:
  $q = 2^{1024} - 2^{960} - 1 + 2^{64} \times \left(\lfloor 2^{894} \times \pi \rfloor + 129093 \right)$,
  $\alpha = 2$.
- Піднесення до степеня в арифметиці класів відрахувань з 1024-бітовим модулем.
- Група еліптичної кривої над кінцевим полем з 2^{155} елементів. Генератор (у шістнадцятковому вигляді): X = 7B, Y = 1C8. Параметри еліптичної кривої (у шістнадцятковому вигляді): A=0, Y=7338F.
- Група еліптичної кривої над кінцевим полем з 2^{185} елементів. Генератор (у шістнадцятковому вигляді): X= 18, Y = D. Параметри еліптичної кривої (у шістнадцятковому вигляді): A=0, Y=1EE9.

Перші три групи являють собою класичний алгоритм Діффі-Хелмана, у якому відбувається піднесення до степеня в арифметиці класів відрахувань. Останні дві групи використовують аналог алгоритму Діффі-Хелмана для еліптичних кривих.

В алгоритмі *Oakley* для захисту від атак відтворення повідомлень застосовуються оказії. Кожна оказія являє собою локально створене псевдовипадкове число. Оказії з'являються у відповідях і шифруються на певних стадіях обміну даними.

З алгоритмом *Oakley* можуть застосовуватися три різних методи автентифікації.

• **Цифрові підписи**. Автентифікація обміну даними здійснюється за допомогою підпису доступного обом сторонам хеш-коду: кожна сторона шифрує хеш-код своїм особистим ключем. Такий хеш-код генерується для важливих параметрів, наприклад, ідентифікаторів користувачів і оказій.

• **Шифрування з відкритим ключем**. Автентифікація обміну даними здійснюється за допомогою шифрування деяких параметрів обміну (наприклад, ідентифікаторів і оказій) з використанням особистого ключа відправника.

• **Шифрування із симетричним ключем**. Для автентифікації обміну

даними може використовуватися шифрування параметрів обміну за симетричною схемою за допомогою певного ключа, який отриманий із застосуванням додаткового механізму.

3.7.3 Приклад обміну *Oakley*

Специфікації *Oakley* включають ряд прикладів обміну, допустимих для даного протоколу. Щоб одержати уявлення про роботу *Oakley*, розглянемо приклад обміну даними, який у документі з описами специфікацій називається енергійним обміном ключами (*aggressive key exchange*).

На рис. 3.11 показана схема протоколу енергійного обміну ключами. На першому кроці ініціатор передає *cookie*, групу, що передбачається використовувати, свій відкритий ключ Діффі-Хелмана для даного обміну. Крім того, сторона-ініціатор вказує алгоритми шифрування з відкритим ключем, хешування та автентифікації, які пропонує використовувати в ході цього обміну. Ще в повідомленні вказуються ідентифікатори ініціатора і респондента та оказія ініціатора для цього обміну. Нарешті сторона-ініціатор приєднує до повідомлення підпис (побудований за допомогою особистого ключа сторони-ініціатора), який охоплює два ідентифікатори, оказію, групу, відкритий ключ Діффі-Хелмана та запропоновані алгоритми.

Одержавши повідомлення, сторона-респондент перевіряє підпис, використовуючи відкритий ключ підпису ініціатора. Потім респондент підтверджує повідомлення, відсилаючи назад *cookie*, ідентифікатор та оказію сторони-ініціатора, а також групу.

$I \rightarrow R$: CKY_I, OK_KEYX, GRP, g^x, EHAO, NIDP, ID_R, N_I, $S_{KI}(ID_I \| ID_R \| N_I \| GRP \| g^x \| EHAO)$

$R \rightarrow I$: CKY_R, CKY_I, OK_KEYX, GRP, g^y, EHAO, NIDP, ID_R, ID_I, N_R, N_I, $S_{KR}(ID_R \| ID_I \| N_R \| N_I \| GRP \| g^y \| g^x \| EHAS)$

$I \rightarrow R$: CKY_I, CKY_R, OK_KEYX, GRP, g^x, EHAO, NIDP, ID_I, ID_R, N_I, N_R, $S_{KI}(ID_I \| ID_R \| N_I \| N_R \| GRP \| g^x \| g^y \| EHAS)$

Позначення:

I – ініціатор;
R – респондент;
CKY_I, CKY_R – рецепти ініціатора та респондента;
OK_KEYX – тип повідомлення обміну ключами;
GRP – ім'я групи Діффі-Хелмана для цього обміну;
$g^x g^y$ – відкриті ключі ініціатора і респондента;
EHAO, EHAS – функції шифрування, хешування, автентифікації, запрпоновані і вибрані;
NIDP – частина повідомлення що залишилась і не підлягає шифруванню;
ID_I, ID_R – ідентифікатори ініціатора та респондента;
N_I, N_R – випадкові помилки ініціатора та респондента для цього обміну;
$S_{KI}[X]$,$S_{KR}[X]$ – підпис для X, що використовує приватний ключ (ключ підпису) ініціатора, респондента;

Рисунок 3.11 – Приклад енергійного обміну ключами Oakley

Крім того, респондент також включає в повідомлення *cookie*, свій відкритий ключ Діффі-Хелмана, обрані алгоритми (з числа запропонованих), ідентифікатор та оказію респондента для цього обміну.

Нарешті респондент приєднує підпис, використовуючи особистий ключ, який підписує два ідентифікатори, дві оказії, групу, два відкритих ключі Діффі-Хелмана та обрані алгоритми.

Одержавши друге повідомлення, ініціатор перевіряє підпис, використовуючи відкритий ключ респондента. Значення оказій у повідомленні підтверджують, що це не відтворення старого повідомлення. Для завершення обміну, ініціатор повинен відіслати повідомлення назад респонденту для підтвердження того, що ініціатор одержав відкритий ключ респондента.

3.7.4 Протокол *ISAKMP*

Протокол захищених зв'язків і керування ключами в *Інтернет* (*Internet Security Association and Key Management Protocol – ISAKMP*). *ISAKMP* забезпечує каркас схеми керування ключами в *Інтернет* і підтримку спеціального протоколу і необхідних форматів процедури узгодження атрибутів захисту.

ISAKMP не змушує використовувати певний конкретний алгоритм обміну ключами, а пропонує набір типів повідомлень, що дозволяють задіяти будь-який подібний алгоритм. *Oakley* є конкретним алгоритмом обміну ключами, призначеним для використання з вихідною версією *ISAKMP*.

Протокол *ISAKMP* визначає процедури і формати пакета, які використовуються для переговорів про створення, зміну або знищення захищених зв'язків. Як частина процесу створення захищеного зв'язку, *ISAKMP* визначає корисний вантаж повідомлень обміну ключами й автентифікації даних. Формати корисного вантажу забезпечують погоджений каркас, що не залежить від конкретного протоколу обміну ключами, алгоритму шифрування або механізму автентифікації.

Повідомлення *ISAKMP* складається із заголовка *ISAKMP* і таких за ним корисних вантажів. Усе це передається за допомогою транспортного протоколу. Будь-яка реалізація підтримує використання *UDP*.

На рис. 3.12, а показаний формат заголовка повідомлення *ISAKMP*, який формується з таких полів.

Cookie **ініціатора** (64 біт). *Cookie* об'єкта, що ініціював процес створення, зміни або знищення захищеного зв'язку.

Cookie **респондента** (64 біт). *Cookie* об'єкта, що відповідає, у першому повідомленні ініціатора залишається порожнім.

Наступний корисний вантаж (8 біт). Указує тип першого корисного вантажу в повідомленні.

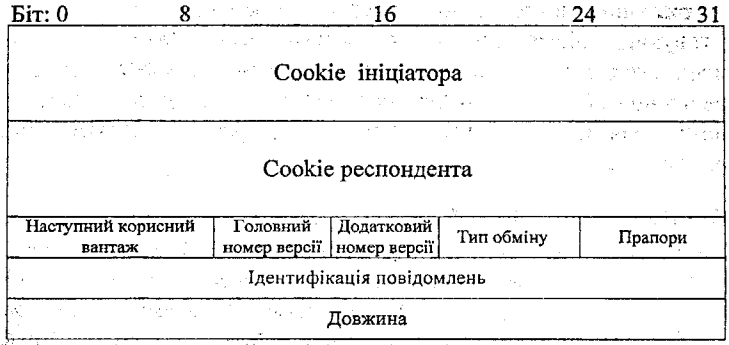
Головний номер версії (4 біт). Указує номер версії *ISAKMP*.

Додатковий номер версії (4 біт). Указує додатковий номер версії.

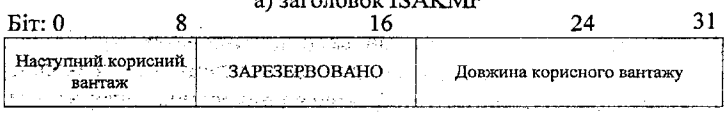
Тип обміну (8 біт). Указує тип обміну.

Прапорці (8 біт). Указують параметри, встановлені для даного обміну *ISAKMP*. На даному етапі в специфікаціях визначено два біти. Біт шифрування встановлюється тоді, коли всі наявні корисні вантажі, які слідують

за заголовком, зашифровані з використанням алгоритму шифрування захищеного зв'язку. Біт фіксації призначений для того, щоб гарантувати, що зашифрований матеріал не був отриманий до створення захищеного зв'язку.



а) заголовок ISAKMP



б) заголовок типового корисного вантажу

Рисунок 3.12 – Формат ISAKMP

Ідентифікатор повідомлення (32 біт). Унікальний ідентифікатор даного повідомлення.

Довжина (32 біт). Довжина всього повідомлення (заголовка і всіх корисних вантажів) у байтах.

3.7.5 Типи корисного вантажу *ISAKMP*

Усі корисні вантажі *ISAKMP* мають заголовки одного типу. Структура такого заголовка показана на рис. 3.12, б. Поле такого корисного вантажу має значення 0, якщо даний корисний вантаж є в повідомленні останнім, інакше значенням поля буде тип наступного корисного вантажу. Значення поля довжини корисного вантажу вказує довжину в байтах відповідного корисного вантажу, включаючи довжину його заголовка.

У таблиці 3.3 наведений список типів корисного вантажу, допустимих для *ISAKMP*, і зазначені поля, або параметри, що складають корисний вантаж кожного з цих типів. Корисний вантаж типу *SA* (захищений зв'язок) служить для того, щоб почати процес створення захищеного зв'язку. У цьому випадку параметр *Domain of Interpretation* ідентифікує область інтерпретації (*DOI*), у рамках якої виконується узгодження. Використання *DOI* протоколу *IPSec* є лише одним прикладом, *ISAKMP* може використовуватися й в інших контекстах. Параметр *Situation* (ситуація) визначає політику захисту для процесу узгодження,

встановлюючи ступінь захисту. Корисний вантаж типу *P* (пропозиція) включає інформацію, яка використовується в ході узгодження атрибутів створюваного захищеного зв'язку, а також указує протокол для цього захищеного зв'язку (*ESP* або *AH*).

Таблиця 3.3 – Типи корисного вантажу *ISAKMP*

Тип	Параметри	Опис
Захищений зв'язок (*SA*)	Область інтерпретації, ситуація	Використовується для узгодження атрибутів захисту і вказанням області інтерпретації і ситуації, у рамках якої виконується таке узгодження
Пропозиція (*P*)	Номер пропозиції, ідентифікатор протоколу, довжина індексу параметрів захисту, число трансформацій, індекс параметрів захисту	Використовується в ході узгодження параметрів створюваного захищеного зв'язку, указує застосовуваний протокол і число трансформацій
Трансформація (*T*)	Номер трансформації, ідентифікатор трансформації, атрибути захищеного зв'язку	Застосовується в ході узгодження параметрів захищеного зв'язку, указує перетворення і відповідні атрибути захищеного зв'язку
Обмін ключами (*KE*)	Дані обміну ключами	Підтримує цілий ряд методів обміну ключами
Ідентифікація (*ID*)	Тип ідентифікації, дані ідентифікації	Призначений для обміну інформацією ідентифікації
Сертифікат (*CERT*)	Кодування сертифіката, дані сертифікації	Використовується для пересилання сертифікатів і іншої зв'язаної із сертифікатами інформації
Запит сертифіката (*CR*)	Число типів сертифікатів, типи сертифікатів, число типів центрів сертифікації, центри сертифікації	Використовується для запитів сертифікатів, указує типи запитуваних сертифікатів і прийнятні центри сертифікації
Хешування (*HASH*)	Дані хешування	Містить дані, які генеруються функцією хешування
Підпис (*SIG*)	Дані підписи	Містить дані, які генеруються функцією цифрового підпису
Оказія (*NONCE*)	Дані оказії	Містить оказію
Повідомлення (*N*)	Область ідентифікації, ідентифікатор протоколу, довжина індексу параметрів захисту, тип повідомлення, індекс параметрів захисту, дані повідомлення	Використовується для передачі даних повідомлення, наприклад, ознаки виникнення помилки
Видалення (*D*)	Область ідентифікації, ідентифікатор протоколу, довжина індексу параметрів захисту, число індексів параметрів захисту, індекс параметрів захисту	Указує захищений зв'язок, який більше не дійсний

Крім того, корисний вантаж цього типу включає індекс параметрів захисту об'єкта-відправника і число трансформацій. Кожна трансформація міститься в корисному вантажі типу *T* (*Transform* – трансформація).

Застосування декількох корисних вантажів типу трансформація дозволяє ініціаторові запропонувати на вибір кілька можливостей, з яких відповідаюча сторона, повинна вибрати одну або відкинути пропозицію.

Корисний вантаж типу *T* визначає захисну трансформацію, що повинна використовуватися для захисту комунікаційного каналу відповідно до зазначеного протоколу. Параметр *Transform #* (номер трансформації) дозволяє ідентифікувати даний корисний вантаж, щоб сторона, що відповідає, могла послатися на цей номер для підтвердження свого рішення використовувати саме цю трансформацію.

Поля *Transform ID* (ідентифікатор трансформації) і *Attributes* (атрибути) визначають трансформацію (наприклад, *3DES* для *ESP* або *HMAC-SHA-1-96* для *AH*) і пов'язані з нею параметри (наприклад, довжину хеш-коду).

Корисний вантаж типу *KE* (обмін ключами) може використовуватися для цілого ряду методів обміну ключами, серед яких *Oakley*, обмін за Діффі-Хелманом та обмін ключами *RSA*, передбачений у *PGP*. Поле *Key Exchange data* (дані обміну ключами) містить дані, необхідні для створення сеансового ключа і залежать від використовуваного алгоритму обміну ключами.

Корисний вантаж типу *ID* (ідентифікатор) призначений для ідентифікації сторін, що з'єднуються, і може використовуватися для перевірки автентичності інформації. Як правило, поле *ID Data* (дані ідентифікації) містить адресу *IPv4* або *IPv6*.

Корисний вантаж типу *CERT* (сертифікат) несе сертифікат відкритого ключа. Поле *Certificate Encoding* (кодування сертифіката) указує тип сертифіката або пов'язану із сертифікатом інформацію (наприклад, сертифікат X.509; сертифікат *PGP*; підписаний ключ *DNS*; сертифікат X.509 – підпис; сертифікат X.509 – обмін ключами; мандати *Kerberos*; список відкликаних сертифікатів (*CRL*); список відкликаних повноважень (*ARL*); сертифікат *SPKI*).

У будь-якій точці обміну *ISAKMP* відправник може розмістити корисний вантаж типу *CR* (запит сертифіката), щоб запросити сертифікат іншої сторони, яка бере участь в обміні. Цей корисний вантаж може містити список декількох типів сертифікатів і декількох центрів сертифікації, що розглядаються як прийнятні.

Корисний вантаж типу *HASH* (хешування) містить дані, які генеруються функцією хешування для деякої частини повідомлення та/або стану *ISAKMP*. За допомогою цього корисного вантажу можна перевірити цілісність даних у повідомленні або автентифікувати об'єкти.

Корисний вантаж типу *SIG* (підпис) містить дані, які генеруються функцією цифрового підпису для деякої частини повідомлення і/або стану *ISAKMP*. Цей корисний вантаж служить для перевірки цілісності даних у повідомленні і може використовуватися в рамках сервісу, що гарантує неможливість відмови від відповідальності.

Корисний вантаж типу *NONCE* (оказія) включає випадкові дані, що забезпечують гарантію виконання процесу обміну в реальному часі і захист його від атак відтворення повідомлень.

Корисний вантаж типу *N* (повідомлення) містить інформацію або про помилку, або про стан, що пов'язується з даним захищеним зв'язком і даним процесом узгодження параметрів захищеного зв'язку.

Визначено такі повідомлення *ISAKMP* про помилки:

Invalid Payload Type (неправильний тип корисного вантажу).

DOI Not Supported (непідтримувана область інтерпретації).

Situation Not Supported (непідтримувана ситуація).

Invalid Cookie (неправильний *Cookie*).

Invalid Major Version (неправильний головний номер версії).

Invalid Minor Version (неправильний додатковий номер версії).

Invalid Exchange Type (неправильний тип обміну).

Invalid Flags (неправильні прапорці).

Invalid Message ID (неправильний ідентифікатор повідомлення).

Invalid Protocol ID (неправильний ідентифікатор протоколу).

Invalid SPI (неправильний індекс параметрів захисту).

Invalid Transform ID (неправильний ідентифікатор трансформації).

Attributes Not Supported (атрибути не підтримуються).

No Proposal Chosen (не обрано ніякої пропозиції).

Bad Proposal Syntax (неправильний синтаксис пропозиції).

Payload Malformed (неправильно сформований корисний вантаж).

Invalid Key Information (неправильна інформація про ключ).

Invalid Cert Encoding (неправильне кодування сертифіката).

Invalid Certificate (неправильний сертифікат).

Bad Cert Request Syntax (неправильний синтаксис запиту сертифіката).

Invalid Cert Authority (неправильний центр сертифікації).

Invalid Hash Information (неправильна інформація хешування).

Authentication Failed (невдало завершилася автентифікація).

Invalid Signature (неправильний підпис).

Address Notification (повідомлення про адресу).

Єдиним дотепер визначеним повідомленням стану *ISAKMP* є стан *Connected* (з'єднаний). На додаток до зазначених повідомлень повідомлення *ISAKMP* використовуються спеціальні повідомлення *DOI*. Для *IPSec* визначаються такі додаткові повідомлення про стан.

Responder-Lifetime. Тривалість життя захищеного зв'язку, обраного відповідаючою стороною.

Replay-Status. Підтверджує позитивне рішення відповідаючої сторони щодо питання про застосування засобів виявлення атак відтворення.

Initial-Contact. Інформує іншу сторону про те, що даний захищений зв'язок є першим, який створюється з віддаленою системою. Одержувач такого повідомлення може знищити всі існуючі захищені зв'язки із

системою-відправником, припускаючи, що система відправника була перезавантажена і більше не має доступу до старих захищених зв'язків.

Корисний вантаж типу *D* (*Delay* – знищення) вказує один або кілька захищених зв'язків, які стали недійсними через те, що відправник видалив їх зі своєї бази даних.

3.7.6 Обмін *ISAKMP*

Протокол *ISAKMP* забезпечує каркас для обміну повідомленнями, будівельними блоками яких служать корисні вантажі. Специфікації вказують п'ять типів обміну, що повинні підтримуватися за замовчуванням (таблиця 3.4). У таблиці *SA* означає корисний вантаж захищеного зв'язку з відповідними корисними вантажами протоколу і трансформації.

Таблиця 3.4 – Типи обміну *ISAKMP*

Повідомлення обміну	Коментар
а) базовий обмін	
(1) I–>R: SA, NONCE	Починається узгодження захищеного зв'язку *ISAKMP*
(2) R–>I: SA, NONCE	Основні параметри захищеного зв'язку погоджені
(3) I–>R: KE, ID_I, AUTH	Ключ згенерований, відповідаюча сторона ідентифікувала ініціатора
(4) R–>I: KE, ID_R, AUTH	Ініціатор ідентифікував відповідаючу сторону, ключ згенерований, установлений захищений зв'язок
б) обмін із захистом ідентифікації сторін	
(1) I–>R: SA	Починається узгодження захищеного зв'язку *ISAKMP*
(2) R–>I: SA	Основні параметри захищеного зв'язку погоджені
(3) I–> R: KE, NONCE	Ключ згенерований
(4) R–>I: KE, NONCE	Ключ згенерований
(5)* R–>I: ID_I, AUTH	Відповідаюча сторона ідентифікувала ініціатора
(6)* I–>R: ID_R, AUTH	Ініціатор ідентифікував відповідаючу сторону, захищений зв'язок установлений
в) обмін тільки даними автентифікації	
(1) I–>R: SA, NONCE	Починається узгодження захищеного зв'язку *ISAKMP*
(2) R–>I: SA, NONCE, ID_R, AUTH	Основні параметри захищеного зв'язку погоджені, відповідаюча сторона ідентифікувала ініціатора
(3) I–>R: ID_I, AUTH	Ініціатор ідентифікував відповідаючу сторону, захищений зв'язок установлений
г) енергійний обмін	
(1) I–>R: SA, KE, NONCE, ID_I	Починається узгодження захищеного зв'язку й обмін ключами *ISAKMP*
(2) R–>I: SA, KE, NONCE, ID_R, AUTH	Відповідаюча сторона ідентифікувала ініціатора, ключ згенерований, основні параметри захищеного зв'язку погоджені
(3)* I–>R: AUTH	Ініціатор ідентифікував відповідаючу сторону, захищений зв'язок установлений
д) інформаційний обмін	
(1)* I–>R: N/D	Повідомлення про помилку, стан або повідомлення про видалення зв'язку
Позначення: * – корисний вантаж після заголовка *ISAKMP* шифрується.	

Базовий обмін дозволяє провести обмін ключами і даними автентифікації одночасно. Це зводить до мінімуму число повідомлень обміну за рахунок відмовлення від захисту ідентифікації сторін. Два перших повідомлення забезпечують обмін *cookies* і створюють захищений зв'язок з погодженими протоколом і трансформаціями; обидві сторони використовують оказії, щоб застрахуватися від атак відтворення повідомлень. У двох повідомленнях, що залишилися, відбувається обмін інформацією, яка стосується ключів, та ідентифікаторами користувачів. А корисний вантаж *AUTH* використовується для автентифікації ключів, що беруть участь в обміні сторін і оказій з перших двох повідомлень.

Обмін із захистом ідентифікації сторін є розширенням базового обміну з метою захисту інформації про сторони, які беруть участь в обміні. Два перших повідомлення створюють захищений зв'язок. Два такі повідомлення здійснюють обмін ключами і включають дві оказії для захисту від атак відтворення. Як тільки стає доступним сеансовий ключ, сторони обмінюються шифрованими повідомленнями, що містять інформацію автентифікації, наприклад цифрові підписи, і, якщо необхідно, сертифікати відкритих ключів.

Обмін тільки даними автентифікації використовується для того, щоб здійснити взаємну автентифікацію без обміну ключами. Два перших повідомлення створюють захищений зв'язок. Крім того, відповідаюча сторона використовує друге повідомлення для того, щоб передати свій ідентифікатор, і застосовує автентифікацію для захисту цього повідомлення. Ініціатор посилає третє повідомлення, щоб передати свій автентифікований ідентифікатор.

Енергійний обмін зводить до мінімуму число повідомлень обміну за рахунок відмови від захисту ідентифікації сторін. У першому повідомленні ініціатор пропонує створити захищений зв'язок з набором запропонованих протоколів і трансформацій. Крім того, ініціатор починає обмін ключами і висилає свій ідентифікатор. В другому повідомленні сторона, що відповідає, указує на прийняття захищеного зв'язку з уже конкретними протоколом і трансформацією, завершує обмін ключами й автентифікує передану інформацію. У третьому повідомленні ініціатор передає результат автентифікації отриманої раніше інформації, шифруючи його за допомогою загального секретного сеансового ключа.

Інформаційний обмін призначений для однобічного пересилання інформації, яка стосується параметрів керування захищеним зв'язком.

4 ПРОБЛЕМИ ЗАХИСТУ *WEB*

4.1 Проблеми безпеки в *Web*

Більша частина можливостей *Інтернет* виявилася повною мірою лише після появи веб-технологій. Мільярди гіперзв'язаних документів перебувають на мільйонах комп'ютерів практично у всіх країнах світу. Це й бібліографії, і журнали, і мультимедійні виставки, і бази даних, і різноманітна ділова та навчальна інформація, а саме: електронні каталоги, енциклопедії, бібліотеки, магазини та виставочні зали.

З огляду на мультимедійні можливості *Web* і простоту роботи з ним, а також мови розробки сценарію, *Java* й елементи керування *Active*, стає зрозумілим, що не випадково *Web* у даний момент є засобом реклами й продажу продукції, який найбільш швидко розвивається. На сучасному етапі розвитку ринку застосування *Web* допомагає бізнесменам гідно конкурувати на ринку. І навпаки, відмова від використання *Web* може принести великі збитки. Більшість фахівців вважає, що фінансові й комерційні можливості *Web* будуть розширюватися разом з удосконаленням технологій захисту й шифрування інформації.

Web, електронна пошта, електронні гроші і безпека – це сучасний аналог каталогів, листів, книг обліку й охоронців. У багатьох сферах бізнесу можливості, запропоновані *Web*, є не основними, а допоміжними. Наприклад, більшість людей продовжує купувати одяг не за каталогом, а безпосередньо прийшовши в магазин – адже там її можна приміряти. Як колись результатом більших можливостей і широкого поширення текстових процесорів у середині 80-х років не стала повна відмова від використання паперу, так і більшість бізнесменів думає, що *Web* є скоріше необхідним доповненням у роботі компанії.

Чим активніше використовується *Web*, тим скоріше виникають проблеми захисту інформації. Оскільки одночасно відзначається постійне зростання комп'ютерного шахрайства. Імовірність шахрайства дуже велика, причому злочинцями можуть стати як співробітники компанії, так і сторонні люди.

Простота доступу й використання *Web* дозволяє індивідуальним користувачам і бізнесменам максимально підвищити ефективність і простоту спілкування за допомогою *Web*. Проте, як і у реальному житті, коли у будинок може вломитися грабіжник й украсти всі цінності, у співтоваристві *Інтернет* існують особистості, чиєю основною метою є крадіжка або руйнування інформації, що перебуває на чужому комп'ютері. Подібно тому, як деякі люди не можуть побороти спокусу, потрапляючи в чужий будинок, так й у багатьох виникає спокуса, коли вони одержують доступ на чийсь комп'ютер. Можливо, ті, хто промишляє в *Інтернет*, навіть небезпечніші за злодіїв і бандитів з реального життя, оскільки тривалий час ніхто і не підозрює про те, що зловмисник краде інформацію.

Сьогодні власний веб-вузол мають не тільки практично всі комерційні організації і більшість державних установ, але і багато приватних осіб. Число індивідуальних і корпоративних користувачів *Інтернет* росте дуже швидко, і практично усі вони мають веб-браузери з графічним інтерфейсом. Як наслідок, комерційні структури з ентузіазмом відносяться до ідеї використання *Web* для організації електронної торгівлі. Однак реальність така, що *Інтернет* і *Web* виявляються досить вразливими з погляду загроз найрізноманітнішого типу. Зіткнення з дійсністю змушує комерційні структури приділяти усе більше уваги засобам забезпечення безпеки *Web*.

По суті, *World Wide Web* можна інтерпретувати як додаток типу клієнт/сервер, що працює в мережі *Інтернет* і внутрішніх мережах підприємств на основі протоколу *TCP/IP*. У такій інтерпретації всі згадувані в даній книзі методи захисту можуть застосовуватись і для захисту *Web*. Однак стосовно *Web* виникає і ряд нових проблем, не розглянутих у контексті комп'ютерної безпеки і захисту звичайних мереж.

Мережа *Інтернет* допускає двосторонній обмін інформацією. На відміну від традиційних засобів поширення інформації, включаючи телетекст, голосову пошту й автоматичне відправлення факсів, *Web* виявляється вразливою до можливості проникнення у веб-вузли і зміни їхнього вмісту через *Інтернет*.

Web усе частіше використовується для доступу до корпоративної інформації та інформації про вироблений компанією продукт, а також для здійснення різноманітних транзакцій. Тому порушення роботи веб-сервера може відбитися не тільки на репутації компанії, але й вилитися в значний матеріальний збиток.

Використовувати веб-браузери зовсім просто, набудовувати і підтримувати веб-сервери відносно неважко, для створення веб-сторінок не потрібно особливої майстерності, але складність відповідного програмного забезпечення винятково висока. Таке складне програмне забезпечення може містити множину потенційних недоліків з погляду гарантії безпеки. Досить коротка історія *Web* насичена прикладами нових і сучасних систем, які стали досить вразливими на практиці.

Веб-сервер можна використовувати як плацдарм для проникнення в будь-яку частину всього комп'ютерного комплексу компанії або організації. Зловмисник може одержати доступ до даних і систем, що не є частиною *Web*, а підключені до сервера в рамках локальної мережі.

Типовими споживачами послуг *Web* є випадкові користувачі, які, зазвичай, не мають спеціальної підготовки (в області питань захисту). Таких користувачів, як правило, не хвилює ризик порушення захисту, вони не мають спеціальних засобів і достатніх знань для того, щоб використовувати ефективні контрзаходи, тому і стають жертвами зловмисників, які використовують різноманітні атаки для досягнення своїх цілей.

4.2 Загрози безпеки *Web*

У табл. 4.1 перераховані основні типи загроз безпеки, що виникають при використанні веб-технологій.

Таблиця 4.1 – Порівняльні характеристики загроз безпеки *Web*

	Загрози	**Наслідок**	**Контрзаходи**
Цілісність	Зміна даних користувача. Впровадження "троянських програм". Зміна інформації в пам'яті. Зміна потоку повідомлень на шляху їхньої передачі	Втрата інформації. Дискредитація комп'ютерної системи. Вразливість до загроз всіх інших типів	Криптографічні контрольні суми
Конфіденційність	Перехоплення даних у мережі. Крадіжка інформації, збереженої сервером. Крадіжка інформації, збереженої клієнтом. Одержання інформації про конфігурацію мережі. Одержання інформації про клієнта, що звертається до сервера	Втрата інформації. Порушення таємниці інформації	Шифрування, проксі-сервери *Web*
Доступність	Припинення сеансу доступу користувача. Перевантаження машини потоком фальшивих спроб доступу. Навмисне переповнення дискового простору або оперативної пам'яті. Ізоляція системи шляхом атак на *DNS*-сервер.	Руйнівні наслідки для системи. Роздратування користувачів. Неможливість виконання роботи користувачами.	Важко запобігти
Автентифікація	Імітація легального користувача порушником. Фальсифікація даних	Неправильне представлення користувачів. Довіра до перекручених даних	Криптографічні технології

Один з підходів до класифікації таких загроз полягає в поділі порушень на пасивні й активні. До пасивних порушень захисту можна віднести перехоплення даних на шляху між браузером і сервером або одержання доступу до закритої інформації на веб-вузлі. До активних порушень відносяться, наприклад, спроби порушника видати себе за іншого користувача, зміна потоку даних між клієнтом і сервером, модифікація інформації, що складає вміст веб-вузла.

Інший підхід до класифікації загроз порушення захисту заснований на використанні інформації про місце їхнього прояву: на веб-сервері, у браузері або потоці даних між браузером і сервером.

4.2.1 Атаки методом міжсайтового скриптінгу

Помилки, пов'язані з міжсайтовим скриптінгом (*Cross site scripting, XSS*), є досить відомими, але як і раніше винятково небезпечними. Їхня

унікальність полягає в тому, що замість безпосередньої атаки сервера вони використовують вразливий сервер як засіб атаки на клієнта. Це може зробити винятково важким відстеження подібних атак, особливо якщо запити не повністю протоколюються.

У багатьох документах розглядається питання вставки *HTML*-коду в вразливий скрипт, але обговорення того, що може бути зроблене за допомогою успішної *XSS*-атаки, як правило, залишається за дужками. Хоча звичайно цього досить для захисту, реальна небезпека *XSS*-атак найчастіше залишається недооціненою. *XSS*-атака звичайно проводиться шляхом конструювання спеціального *URL*, який зловмисник підсуває своїй жертві.

Можна провести аналогію між *XSS*-атакою та переповненням буфера. В обох випадках існують дві можливості для здійснення успішної атаки: вставка сміття або перехід в іншу точку. Вставка сміття при переповненні буфера звичайно приводить до атаки на відмову в обслуговуванні. У випадку *XSS*-атаки, вона дозволяє атакуючому відобразити довільну інформацію й припинити виведення оригінальної веб-сторінки. Що стосується переходу в іншу точку при переповненні буфера, він звичайно відбувається в деяку область пам'яті, у якій розташований код, який дозволяє захопити контроль над виконанням програми. У випадку *XSS*, зловмисник перенаправляє жертву на деяку веб-сторінку (як правило, вона перебуває під контролем зловмисника), перехоплюючи поточну сесію. Найпростіша сторінка, вразлива до подібних атак, виглядає так:

```
<?php echo "Hello, {$HTTP_GET_VARS['name']}!"; ?>
```

При відкритті сторінки, змінна *name*, яка відправляється методом *GET*, виводиться безпосередньо в її тілі, з усіма метасимволами. Передавши як параметр "*Gavin Zuchlinski*", ми одержимо коректний варіант відображення (рис. 4.1):

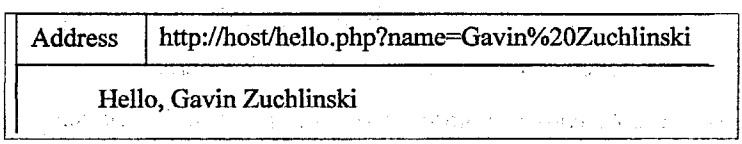


Рисунок 4.1 – Вікно браузера після успішного виконання атаки текстом

Передавши в параметрі *HTML*-код, можна домогтися несподіваного результату (рис. 4.2).

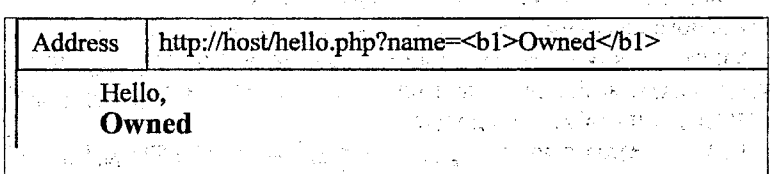


Рисунок 4.2 – Вікно браузера після успішного виконання атаки HTML-кодом

Введення не перевіряється серверним скриптом перед тим, як передати результат браузеру, і це дозволяє вставити у вразливу сторінку довільний код користувача. Іноді введення не обробляється скриптом безпосередньо, а вставляється у файл або базу даних, і після цього забирається звідти для вставки в сторінку. Традиційним місцем для *XSS*-помилок є сторінки, що виводять усілякі підтвердження (наприклад, пошукові скрипти дублюють введенні користувачем дані перед результатами пошуку), і сторінки з повідомленнями про помилки, які повідомляють користувачеві, що саме він увів не так. В останньому випадку (а іноді й у першому) для атаки часто досить закрити вміст текстового поля символами " та >. Лапки закривають параметр *value*, а > – увесь тег.

4.2.2 Фальсифікація *cookie*

Cookie є рішенням однієї з проблем *HTTP* специфікації. Ця проблема полягає в змінності з'єднання між клієнтом і сервером, при сесії *FTP* або *Telnet*, тобто для кожного документа (або файлу) при передачі за протоколом *HTTP* посилається окремий запит. Включення *cookie* в протокол *HTTP* дало часткове рішення цієї проблеми. *Cookie* – це певне значення в текстово-цифровому вигляді, яке сервер передає браузеру. Браузер буде зберігати цю інформацію й передавати її серверу з кожним запитом як частину заголовка *HTTP*. Одні значення *cookie* можуть зберігатися тільки протягом однієї сесії й знищуються після закриття браузера. Інші, встановлені на деякий період часу, записуються у файл. От як виглядає схема роботи хосту та клієнта через *Інтернет* (рис. 4.3):

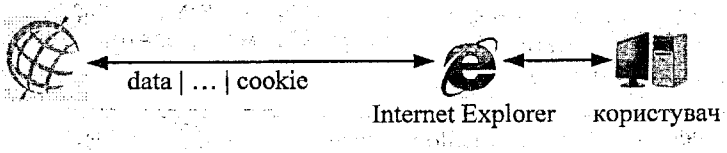


Рисунок 4.3 – Схема клієнт-серверної взаємодії

Самі по собі *cookies* нічого не можуть робити. Однак сервер може зчитувати інформацію, що утримується в *cookies* та на підставі її аналізу робити ті або інші дії.

Клієнт має такі обмеження для *cookies*:

- усього може зберігатися до 300 значень *cookies*;
- кожен *cookie* не може перевищувати 4 Кбайт;
- з одного сервера або домену може зберігатися до 20 значень *cookies*.

У випадку, якщо *cookie* приймає нове значення при наявному вже в браузері *cookie* зі збіжними даними, старе значення затирається новим. В інших випадках нові *cookies* додаються.

Використання *cookie* – ефективне рішення збереження інформації користувача. Багато хто вважає їх небезпечними, нібито вони можуть

заразити комп'ютер вірусом або украсти їхній пароль підключення до *Інтернет*. Як не дивно це твердження є помилковим. Але вважати *cookie* безпечними теж не можна. Для прикладу можна навести кілька прикладів їхнього використання та їх "небезпеки".

1. При використанні пошти з веб-інтерфейсом. Щоб не вводити пароль при кожному вході на поштову скриньку, можна поставити галочку біля надпису "Зберегти пароль". Внаслідок чого інформація з логіном і паролем зберігається в *cookie* і при кожному вході на пошту пароль і логін встановлюється автоматично.

2. Схожий приклад можна привести й з *Інтернет*-магазинами. Заповнивши форму з даними про вашу кредитну картку вони зберігаються й при такій покупці вам не прийде заповнювати всі заново.

Це зручно з погляду користувача, але про безпеку даної технології не може бути й мови. Хоча деякі методи захисту все-таки використовуються.

- Інформація з *cookie* одного домену другого рівня не може бути прочитана іншими доменами.
- Якщо документ кешується, то інформація про *cookie* не кешується.
- Інформація *cookie* може передаватися за допомогою протоколу *SSL*.

Підмінити *cookie* набагато простіше, ніж їх прочитати. Хоча якщо мова йде про віддалений комп'ютер, то завдання стають рівносильними. Справа в тому, що підміна – це просте відправлення *cookie* на сервер, а читання дозволене тільки серверу. Для реалізації читання використається так звана технологія *Cross Site Scripting(CSS* або *XSS*, названий так, щоб уникнути плутанини з *CSS, Cascade Style Sheets*).

Нехай певний користувач має пошту на *www.mail.ru*. Поставивши галочку "Зберегти пароль", всі його дані зберігаються в *cookies*. Тепер є два варіанти розвитку сценарію: якщо є фізичний або віддалений доступ до комп'ютера та якщо доступу немає.

Для першого варіанта нам просто потрібно запустити експлойт. Для другого варіанта потрібний спосіб змусити користувача запустити скрипт. Саме тут і виникає питання ефективності даного методу. При відсутності фізичного або віддаленого доступу до комп'ютера надійніше використати вразливість *Інтернет*-ресурсу. І навпаки. Схема "захоплення" *cookie* має вигляд зображений на рис. 4.4.

При необхідності ці дані можна перенести на інший комп'ютер або змінити. Робиться це в такий спосіб:

```
<script>
onload=function () {
setTimeout(
function () {
document.getElementById("oVictim").Document.cookie="name=value";},
100
);
}
</script>
<iframe src="http://www.mail.ru" id="oVictim"></iframe>
```

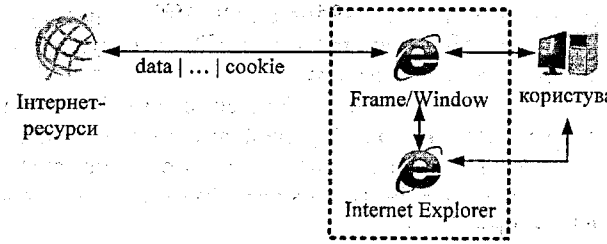


Рисунок 4.4 – Схема „захоплення" cookies

Cookie можна не змінювати, а надіслати запит на сервер із уже зміненими даними:

```
GET /URL HTTP/1.0
Set-Cookie:name=value;EXPIRES=date;DOMAIN=domain_name;PATH=path;SECURE
```

4.2.3 Способи захисту потоку даних у *Web*

Існує кілька підходів до забезпечення захисту даних у *Web*. Усі вони схожі з погляду наданих можливостей і, до деякої міри, з погляду використовуваних механізмів захисту, але розрізняються за областями застосування і розміщенням відповідних засобів захисту в стеці протоколів *TCP/IP*.

Ці розходження схематично показані на рис. 4.5.

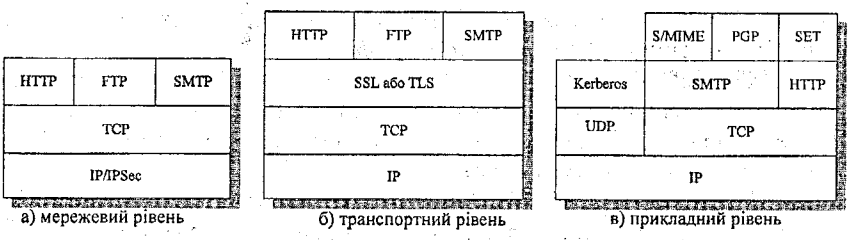


Рисунок 4.5 – Розміщення засобів захисту в стеці протоколів TCP/IP

Один з методів захисту даних у *Web* полягає у використанні протоколу *IPSec* (рис. 4.5, а). Перевага використання *IPSec* у тому, що цей протокол прозорий для кінцевого користувача і програмних додатків та забезпечує універсальне рішення. Крім того, протокол *IPSec* включає засоби фільтрації, які дозволяють використовувати її тільки для тієї частини потоку даних, для якої це дійсно необхідно.

Іншим (певною мірою універсальним) рішенням є розташування засобів захисту відразу над протоколом *TCP* (рис. 4.5, б). Прикладом сучасної реалізації такого підходу є стандарт *SSL* (*Secure Socket Layer* – протокол захищених сокетів) та стандарт *TLS* (*Transport Layer Security* – протокол захисту транспортного рівня). На цьому рівні для практичної

реалізації даного підходу існує дві можливості. Загальним рішенням є впровадження засобів *SSL* (або *TLS*) у набір відповідних протоколів, що забезпечує прозорість засобів захисту для додатків. У той же час засоби *SSL* можна вбудовувати також у прикладні програми. Наприклад, браузери *Netscape* і *Microsoft Internet Explorer*, а також більшість веб-серверів мають вбудовану підтримку *SSL*.

Різні засоби захисту можуть вбудовуватися у програмні додатки, як показано на рис. 4.5, в. Перевага даного підходу полягає в тому, що відповідні засоби захисту можуть бути надбудовані оптимальним чином в залежності від вимог конкретного програмного додатка. У контексті безпеки *Web* важливим прикладом реалізації такого підходу є протокол *SET* (*Secure Electronic Transaction* – протокол захисту електронних транзакцій).

4.3 Протоколи *SSL* і *TLS*

Протокол *SSL* був запропонований компанією *Netscape*. Даний протокол створювався в умовах відкритого обговорення і з урахуванням пропозицій користувачів, після чого він був опублікований у вигляді проекту стандарту захисту даних у *Інтернет*. Згодом після досягнення консенсусу протокол був висунутий на роль стандарту для *Інтернет*, і в рамках *IETF* була сформована робоча група *TLS*, задачею якої є створення загальноприйнятого стандарту.

4.3.1 Архітектура *SSL*

Протокол *SSL* покликаний забезпечити можливість надійного захисту наскрізної передачі даних з використанням протоколу *TCP*. Строго говорячи, *SSL* являє собою не один протокол, а два рівні протоколів, як показано на рис. 4.6.

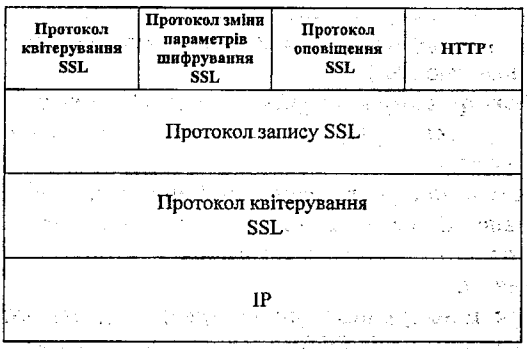


Рисунок 4.6 – Стек протоколів SSL

Протокол *Запису SSL* (*SSL Record Protocol*) забезпечує базовий набір засобів захисту, застосовуваних протоколами більш високих рівнів. Ці за-

соби, зокрема, може використовувати протокол передачі гіпертекстових файлів (*HTTP*), призначений для забезпечення обміну даними при взаємодії клієнтів і серверів *Web*. Частиною *SSL* вважаються і три протоколи більш високого рівня: протокол квітерування встановлення зв'язку (*Handshake Protocol*), протокол зміни параметрів шифрування (*Change Cipher Spec Protocol*) і протокол оповіщення (*Alert Protocol*).

Робота протоколу *SSL* описується в термінах двох важливих понять – сеансу *SSL* і з'єднання *SSL*.

З'єднання (*connection*). З'єднанням називається транспорт (у термінах моделі *OSI*), що забезпечує сервіс деякого необхідного типу. У *SSL* такі з'єднання являють собою рівноправні відносини між вузлами. З'єднання є тимчасовими. Кожне з'єднання асоціюється тільки з одним сеансом.

Сеанс (*session*). Сеанс *SSL* – це зв'язок між клієнтом і сервером. Сеанси створюються протоколом *Квітерування SSL* (*SSL Handshake Protocol*). Сеанс визначає набір параметрів криптографічного захисту, що можуть використовуватися декількома з'єднаннями. Сеанси дозволяють уникнути необхідності ведення переговорів про встановлення параметрів захисту для кожного нового з'єднання.

Між будь-якою парою сторін (наприклад, між *HTTP*-додатками клієнта і сервера) можна установити багато захищених з'єднань. Теоретично між сторонами можна установити і декілька одночасно існуючих сеансів, але на практиці така можливість не використовується.

Кожен сеанс характеризується тим або іншим станом. Наприклад, після початку сеансу встановлюється робочий стан (*operating state*) для читання і запису (тобто одержання і відправлення інформації). А під час роботи протоколу квітерування *SSL* створюється стан чекання (*pending state*) читання і запису. Після успішного завершення роботи протоколу квітерування *SSL* стан чекання переходить у робочий стан.

Стан сеансу визначається такими параметрами (визначення даних згідно зі специфікацією *SSL*).

Ідентифікатор сеансу (*session identifier*). Довільна послідовність байтів, яка генерується сервером для ідентифікації стану активного або поновлюваного сеансу.

Сертифікат вузла (*peer certificate*). Сертифікат *X509.v3* сторони, яка бере участь у сеансі. Цей елемент стану може бути порожнім.

Метод стиснення (*compression method*). Алгоритм стиснення даних перед шифруванням.

Параметри шифрування (*cipher spec*). Задають алгоритм шифрування даних (наприклад, *null*, *DES* і т.п.) і алгоритм хешування (наприклад, *MD5* або *SHA-1*), які використовуються для обчислення значень *MAC*. Крім того, визначають криптографічні атрибути типу довжини хеш-коду.

Головний ключ (*master secret*). 48-байтовий секретний ключ, який використовується клієнтом і сервером.

Прапорець поновлення (*is resumable*). Ознака, що дозволяє ініціалізацію нових з'єднань у рамках даного сеансу.

Стан з'єднання визначається такими параметрами.

Випадковий ідентифікатор сервера і клієнта. Послідовність байтів, яка генерується для ідентифікації кожного з'єднання між сервером і клієнтом.

Ключ запису сервера *MAC* (*Message Authentication Code* – код автентичності повідомлення). Секретний ключ, який застосовується сервером при обчисленні значень *MAC* для даних, які їм відправляються.

Ключ запису клієнта *MAC*. Секретний ключ, який застосовується клієнтом при обчисленні значень *MAC* для даних, які їм відправляються.

Ключ запису сервера. Ключ традиційної схеми шифрування, який використовується сервером для шифрування даних і клієнтом для дешифрування.

Ключ запису клієнта. Ключ традиційної схеми шифрування, який використовується клієнтом для шифрування даних і сервером для дешифрування.

Вектори ініціалізації. При блоковому шифруванні в режимі *CBC* (режим зчеплення шифрованих блоків) з кожним ключем пов'язується вектор ініціалізації (*IV*). Перший раз це поле ініціалізується протоколом квітерування *SSL*, а потім для кожного такого запису як вектор ініціалізації використовується збережений поточний шифрований блок.

Порядкові номери. Кожна зі сторін самостійно нумерує повідомлення, які відправляються та отримуються при кожному з'єднанні. Коли одна зі сторін одержує або відправляє повідомлення про зміну параметрів шифрування, відповідний порядковий номер встановлюється рівним 0. Порядкові номери не можуть перевищувати значення $(2^{64} - 1)$.

4.3.2 Протокол запису *SSL*

Протокол *Запису SSL* (*SSL Record Protocol*) забезпечує підтримку двох сервісів для з'єднань *SSL*.

Конфіденційність. Протокол *Квітерування SSL* визначає загальний для клієнта і сервера секретний ключ, який буде використовуватися алгоритмом традиційної схеми для шифрування даних, переданих по протоколу *SSL*.

Цілісність повідомлень. Крім забезпечення конфіденційності, протокол *Квітерування SSL* визначає загальний секретний ключ для обчислення значень *MAC*.

На рис. 4.7 показана загальна схема роботи протоколу *Запису SSL*.

Одержавши повідомлення для відправлення, протокол спочатку фрагментує дані, розбиваючи їх на блоки підходящого розміру, при необхідності виконує стиснення даних, застосовує алгоритм обчислення *MAC*, шифрує дані, додає заголовок і передає отримані пакети сегментові *TCP*. Прийняті дані дешифруються, перевіряються, розпаковуються, збираються знову з фрагментів і передаються додаткам більш високого рівня.

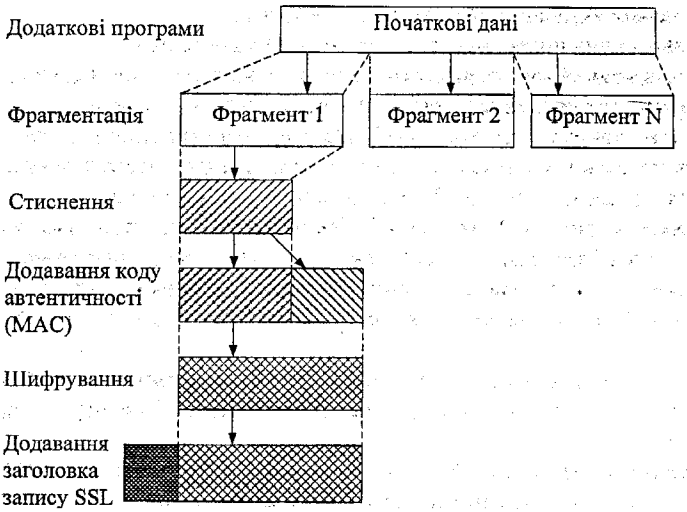


Рисунок 4.7 – Схема роботи протоколу Запису SSL

Першим кроком є фрагментація. Повідомлення, отримане від додатка більш високого рівня, розділяється на блоки розміром не більш 2^{14} байт (16384 байт). Потім, при необхідності, застосовується стиснення. Таке стиснення повинно відбуватися без втрат і не повинне збільшувати розмір блока більше ніж на 1024 байт.

Таким кроком є обчислення коду автентичності повідомлення (значення *MAC*) для даних. Для цього використовується загальний секретний ключ. Обчислення виконуються за такою схемою.

```
hash(MAC_write_secret || pad_2 ||
hash(MAC_write_secret || pad_1 || seq_num || SSLCompressed.type ||
SSLCompressed.length || SSLCompressed.fragment)),
```

де || – операція конкатенації;

MAC_write_secret – загальний секретний ключ;

Hash – криптографічний алгоритм хешування (*MD5* або *SHA-1*);

pad_1 – байт 0x36 (*ООН ВОНО*), повторений 48 разів (384 біт) при використанні *MD5* або 40 разів (320 біт) при використанні *SHA-1*;

pad_2 – байт *Ox5C* (0101 1100), повторений 48 разів для *MD5* або 40 разів для *SHA-1;*

seq_num – порядковий номер даного повідомлення;

SSLCompressed.type – протокол більш високого рівня, який використовується для обробки даного фрагмента;

SSLCompressed.length – довжина стиснутого фрагмента;

SSLCompressed.fragment – стиснутий фрагмент (якщо стиснення не

застосовувалось, то фрагмент у вигляді відкритого тексту).

Цей алгоритм дуже схожий на алгоритм *HMAC*. Розходження полягає в тому, що в *SSL* заповнювачі пов'язуються операцією конкатенації, а в *HMAC* – операцією *XOR* (виключне "АБО").

Потім повідомлення разом з доданим до нього значенням *MAC* шифрується з використанням симетричної схеми шифрування. Шифрування теж не повинне збільшувати довжину блока більше ніж на 1024 байт, тому загальний розмір блока не може перевищити (2^{14}+2048) байт. Допускається застосування алгоритмів шифрування вказаних в табл. 4.2.

Таблиця 4.2 – Алгоритми шифрування допустимі в *SSL*

Блокове шифрування		Потокове шифрування	
Алгоритм	Довжина ключа	Алгоритм	Довжина ключа
IDEA	128	*RC4-40*	40
RC2-40	40	*RC4-128*	128
DES-40	40		
DES	56		
3DES	168		
Fortezza	80		

Алгоритм *Fortezza* може використовуватися в схемах шифрування мікропроцесорних карт (*smart card*). У випадку застосування алгоритмів потокового шифрування шифруються тільки повідомлення і додане до нього значення *MAC*. Значення *MAC* обчислюється перед шифруванням і, таким чином, шифрується разом з відкритим текстом або стиснутим відкритим текстом повідомлення.

При використанні алгоритмів блокового шифрування після значення *MAC* може знадобитися додати заповнювач. При цьому безпосередньо за байтами заповнювача слідує 1-байтове значення, що вказує загальну довжину заповнювача. Для загальної довжини заповнювача вибирається найменше зі значень, при якому довжина блока даних, які підлягають шифруванню (відкритий текст + *MAC* + заповнювач), виявляється кратною довжині блока шифру. Розглянемо приклад, коли довжина відкритого тексту дорівнює 58 байт, довжина значення *MAC* – 20 байт (алгоритм *SHA-1*), а алгоритм шифрування (наприклад *DES*) використовує блоки довжиною 8 байт. З урахуванням байта, який вказує довжину заповнювача (*padding.length*), загальний розмір блока складе 79 байт. Щоб розмір блока був кратний 8, у даному випадку необхідно додати один байт заповнювача.

Завершальним кроком у роботі протоколу запису *SSL* є створення заголовка, що складає з таких полів.

Тип вмісту (8 біт). Визначає протокол верхнього рівня, за допомогою якого повинен оброблятися даний фрагмент.

Головний номер версії (8 біт). Указує головний номер версії застосовуваного протоколу *SSL*.

Додатковий номер версії (8 біт). Указує додатковий номер версії застосовуваного протоколу *SSL*.

Довжина стиснутого фрагмента (16 біт). Довжина в байтах даного фрагмента відкритого тексту (або стиснутого фрагмента, якщо використовується стиснення). Максимально допустиме значення дорівнює (2^{14}+2048).

Для типу вмісту передбачене використання значень *handshake, change_cipher_spec, alert*, і *application_data*. Перші три значення визначають протоколи стека *SSL* і будуть описані нижче. Між додатками, що можуть використовувати *SSL* (наприклад *HTTP*), розходжень не робиться: для *SSL* створені такими додатками дані мають однакове значення.

Формат запису, одержуваного на виході *SSL*, показаний на рис. 4.8.

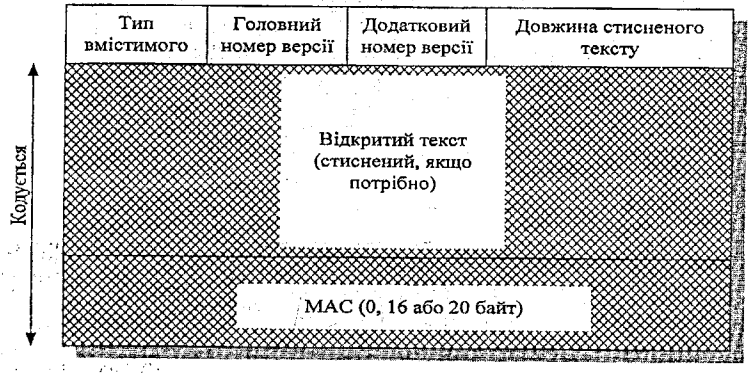


Рисунок 4.8 – Формат запису SSL

4.3.3 Протокол *Зміни параметрів шифрування*

Протокол *Зміни параметрів шифрування* (*Change Cipher Spec Protocol*) є найпростішим із трьох протоколів верхнього рівня в стеці протоколів *SSL*. Протокол *Зміни параметрів шифрування* генерує однобайтове повідомлення, що містить значення 1 (рис. 4.9, а). Єдиною метою цього повідомлення є вказівка копіювати параметри стану чекання в поточний стан, у результаті чого обновляється комплект шифрів, які використовуються для даного з'єднання.

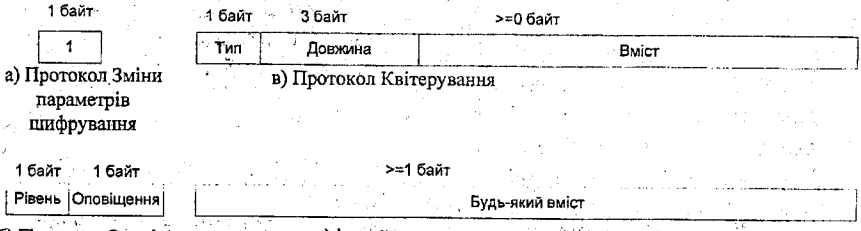


Рисунок 4.9 – Формат корисного вантажу протоколу запису SSL

4.3.4 Протокол *Попередження*

Протокол *Попередження* (*Alert Protocol*) призначений для передачі іншій стороні, яка бере участь в обміні даними, повідомлень, які стосуються роботи *SSL*. Як і дані будь-якого іншого програмного додатка, який використовує *SSL*, повідомлення протоколу стискуються і шифруються відповідно до параметрів поточного стану.

Будь-яке повідомлення, яке генерується в рамках даного протоколу, складається з двох байтів (рис. 4.9, б). Перший байт містить значення, що позначає відповідно рівень попередження (1) або рівень критичної помилки (2). Якщо зазначено рівень критичної помилки, протокол *SSL* негайно розриває з'єднання. Інші з'єднання даного сеансу можуть продовжувати існувати, але установити нове з'єднання для даного сеансу буде вже неможливо. Другий байт містить код, що позначає конкретний зміст повідомлення. Перелічимо повідомлення, що вказують на критичну помилку (відповідно до специфікацій *SSL*).

unexpected_message. Отримано непридатне для обробки повідомлення.

bad_record_mac. Отримано неправильне значення *MAC*.

decompression_failure. Функція розпакування стиснутих даних одержала неправильні вхідні дані (тобто дані неможливо розпакувати або довжина розпакованих даних більша максимально припустимої).

handshake_failure. Відправник не зміг погодити з одержувачем параметри захисту на основі наявних можливостей.

illegal_parameter. Значення поля в повідомленні квітерування виходить за рамки припустимого діапазону або не узгоджується зі значеннями інших полів.

Крім зазначених, є такі повідомлення.

close_notify. Сповіщає одержувача про те, що відправник більше не буде відправляти повідомлення з використанням даного з'єднання. Щоб коректно закрити режим запису будь-якого з'єднання, відправити повідомлення *close_notify* перед закриттям з'єднання повинна кожна зі сторін.

no_sertificate. Може відправлятися у відповідь на запит про наявність сертифіката, якщо в сторони, що відповідає, відповідного сертифіката немає.

bad_sertificate. Отриманий сертифікат ушкоджений (тобто містить підпис, верифікацію якого виконати не удалося).

unsupported_sertificate. Тип отриманого сертифіката не підтримується.

sertificate_revoked. Сертифікат був відкликаний об'єктом, який його підписав.

sertificate_expired. Термін придатності сертифіката минув.

sertificate_unknown. При обробці сертифіката виникли якісь інші проблеми, унаслідок чого його використання виявляється неможливим.

4.3.5 Протокол *Квітерування*

Найскладнішим у стеці протоколів *SSL* є протокол *Квітерування* (*Handshake Protocol*). Цей протокол дозволяє серверові і клієнтові викона-

ти взаємну автентифікацію, а також погодити алгоритми шифрування, обчислення *MAC* і криптографічні ключі, які будуть застосовуватися для захисту даних, що пересилаються протоколом *SSL*. Протокол *Квітерування* повинен використовуватися до початку пересилання даних прикладних програм.

У рамках протоколу *Квітерування* генерується кілька повідомлень, якими обмінюються клієнт і сервер. Усі вони мають формат, показаний на рис. 4.9, в. Будь-яке таке повідомлення містить три таких поля.

Тип (1 байт). Вказує один з 10 припустимих типів повідомлення. Припустимі типи повідомлень наведені в табл. 4.2.

Довжина (3 байт). Довжина повідомлення в байтах.

Вміст (>= 1 байта). Параметри, що пов'язуються з повідомленням даного типу (див. табл. 4.2).

Таблиця 4.2 – Типи повідомлень протоколу *Квітерування*

Тип повідомлення	Параметри
hello_request	Немає
client_hello	Версія, випадкові значення, ідентифікатор сеансу, комплект шифрів, метод стиснення
server_hello	Версія, випадкові значення, ідентифікатор сеансу, комплект шифрів, метод стиснення
certificate	Ієрархія сертифікатів X.509v3
server_key_exchange	Параметри, підпис
certificate_request	Тип сертифіката, назви уповноважених об'єктів
server_done	Немає
certificate_verify	Підпис
client_key_exchange	Параметри, підпис
Finished	Хеш-код

На рис. 4.10 показана схема обміну повідомленнями при встановленні логічного з'єднання між клієнтом і сервером.

Процес обміну складається з таких чотирьох основних етапів.

4.3.6 Визначення характеристик захисту (Етап 1)

На цьому етапі наводиться ініціалізація логічного з'єднання і визначаються пов'язані з ним характеристики захисту. Процес ініціюється клієнтом, що відправляє серверові повідомлення *client_hello* з такими параметрами.

Версія. Найвищий номер версії *SSL*, підтримуваний клієнтом.

Випадкове значення. Генерована клієнтом випадкова структура, яка містить 32-бітовий штамп дати/часу і 28 байт, отриманих за допомогою захищеного генератора випадкових чисел. Ці значення використовуються як оказії під час обміну ключами з метою захисту від атак відтворення.

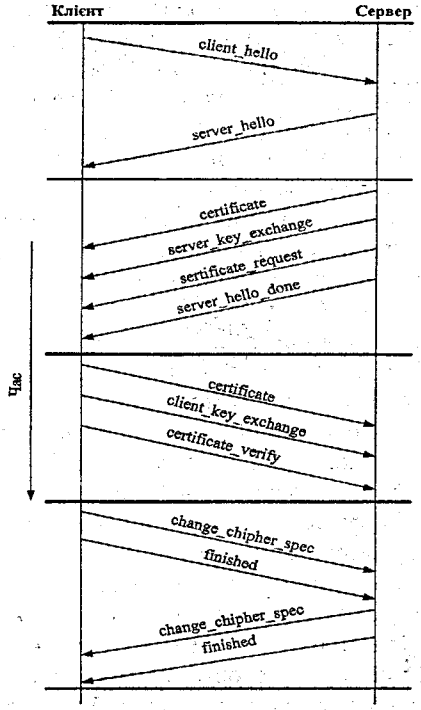


Рисунок 4.10 – Схема роботи протоколу квітерування

Ідентифікатор сеансу. Ідентифікатор змінної довжини для даного сеансу. Ненульове значення говорить про намір клієнта обновити параметри наявного з'єднання або створити нове з'єднання в рамках того ж сеансу. Нульове значення говорить про те, що клієнт має намір створити нове з'єднання в новому сеансі.

Комплект шифрів. Список, що містить набори криптографічних алгоритмів, підтримуваних клієнтом, перераховані в порядку зменшення їхнього пріоритету. Кожен елемент списку (кожен комплект шифрів) визначає алгоритм обміну ключами і специфікації шифру (*CipherSpec*).

Метод стиснення. Список методів стиснення, підтримуваних клієнтом.

Після відправлення повідомлення *client_hello* (вітання клієнта) клієнт очікує від сервера повідомлення *server_hello* (вітання сервера), що містить ті ж параметри, що і повідомлення *client_hello*. Для повідомлення *server_hello* діють такі угоди. Поле *Version* (версія) містить номер нижчої версії, підтримуваної клієнтом, і вищої версії, підтримуваної сервером. Поле *Random* (випадкове значення) генерується сервером і не пов'язано зі значенням поля *Random* у повідомленні клієнта. Якщо значення поля

SessionID (ідентифікатор сеансу) у повідомленні клієнта було відмінним від нуля, те ж значення використовує і сервер. В іншому випадку поле *SessionID* у повідомленні сервера містить значення ідентифікатора для нового сеансу. У полі *CipherSuite* (комплект шифрів) визначається один комплект шифрів, обраний сервером зі списку, запропонованого клієнтом. У полі *Compression* (стиснення) визначається метод стиснення, обраний сервером зі списку, запропонованого клієнтом.

Першим елементом поля *CipherSuite* визначається метод обміну ключами (тобто метод, на основі якого буде відбуватися обмін ключами для схем традиційного шифрування й обчислення значень *MAC*). Протокол *SSL* підтримує такі методи обміну ключами.

RSA. Секретний ключ шифрується за допомогою відкритого ключа *RSA* одержувача. Для цього відправникові повинен бути доступний сертифікат відкритого ключа одержувача.

Метод Діффі-Хелмана з фіксованими параметрами (*Fixed Diffie-Hellman*). Метод обміну ключами за схемою Діффі-Хелмана, при якому сертифікат сервера містить відкриті параметри алгоритму Діффі-Хелмана, підписані центром сертифікації. Іншими словами, відповідний сертифікат містить параметри відкритого ключа алгоритму Діффі-Хелмана. Клієнт повідомляє свої параметри відкритого ключа алгоритму Діффі-Хелмана або в сертифікаті, при автентифікації клієнта, або в повідомленні обміну ключами.

Метод Діффі-Хелмана з одноразовими параметрами (*Ephemeral Diffie-Hellman*). Цей метод застосовується для створення "ефемерних" (тимчасових, одноразових) секретних ключів. У цьому випадку сторони обмінюються відкритими ключами Діффі-Хелмана, підписаними особистим ключем (*RSA* або *DSS*) відправника. Одержувач для перевірки підпису може використовувати відповідний відкритий ключ. Для автентифікації відкритих ключів застосовуються сертифікати. Цей метод є найбезпечнішим із трьох зазначених тут варіантів методу Діффі-Хелмана, тому що в результаті отримують тимчасовий автентифікований ключ.

Анонімний метод Діффі-Хелмана (*Anonymous Diffie-Hellman*). Припускає використання базового алгоритму Діффі-Хелмана, але автентифікація не виконується. Іншими словами, кожна зі сторін відправляє свої відкриті параметри для алгоритму Діффі-Хелмана іншій стороні без автентифікації. Даний підхід виявляється вразливим до атак "впровадження посередника", коли з обома сторонами обмін ключами проводить зловмисник.

Fortezza. Метод, використовуваний у схемі шифрування *Fortezza*.

Визначає характеристики захисту, включаючи номер версії протоколу, ідентифікатор сеансу, комплект шифрів, метод стиснення і вихідні випадкові числа.

Сервер може передати сертифікат, повідомлення обміну ключами і запит сертифіката. Сервер сигналізує про закінчення фази привітального повідомлення.

Клієнт передає сертифікат, якщо він був запитаний. Клієнт передає повідомлення обміну ключами. Клієнт може передати повідомлення верифікації сертифіката.

Зміна комплекту шифрів і завершення роботи протоколу квітерування.

Потім слідує поле *CipherSpec* (параметри шифрування), яке складається з таких полів.

CipherAlgoritm. Алгоритм шифрування: кожного з раніше згадуваних алгоритмів (*RC4, RC2, DES, 3DES, DES40, IDEA, Fortezza*).

MAC Algorithm. Алгоритм обчислення *MAC*: *MD5* або *SHA-1*.

CipherType. Тип шифру: *Stream* (потоковий) або *Block* (блоковий).

IsExportable. Ознака можливості експорту: *True* або *False*. Довжина хеш-коду; 0, 16 (для *MD5*) або 20 (для *SHA-1*) байт.

Key Material. Параметри обчислення ключів: послідовність байтів, що містять дані, які використовуються при генеруванні ключів запису.

IV Size. Довжина вектора ініціалізації для шифрування в режимі *CBC* (режим зчеплення шифрованих блоків).

4.3.7 Автентифікація й обмін ключами сервера (Етап 2)

Даний етап починається з відправлення сервером його сертифіката, якщо потрібна автентифікація сервера. Повідомлення, що відправляється, містить сертифікат *X.509* або ієрархію таких сертифікатів. Повідомлення *certificate* (сертифікат) потрібно для кожного з вказаних вище методів обміну ключами, крім анонімного методу Діффі-Хелмана. Зверніть увагу на те, що при використанні методу Діффі-Хелмана з фіксованими параметрами повідомлення сертифіката відіграє роль повідомлення обміну ключами, оскільки в ньому містяться запропоновані сервером відкриті параметри алгоритму Діффі-Хелмана.

Потім може бути відправлене повідомлення *server_key_exchange* (обмін ключами сервера). Відправляти таке повідомлення не потрібно в двох випадках: коли сервер відправив сертифікат для методу Діффі-Хелмана з фіксованими параметрами, та коли пропонується використовувати схему обміну ключами *RSA*. Повідомлення *server_key_exchange* необхідно тоді, коли використовуються такі схеми.

Анонімний метод Діффі-Хелмана. Повідомлення містить два глобальних значення, необхідні для використання методу Діффі-Хелмана (деяке просте число і його первісний корінь), а також відкритий ключ сервера для алгоритму Діффі-Хелмана.

Метод Діффі-Хелмана з одноразовими параметрами. Повідомлення містить такі ж три параметри, як і в анонімному методі Діффі-Хелмана, та підпис для цих параметрів.

Обмін ключами за схемою RSA, коли сервер має ключ RSA тільки для підпису. У цьому випадку в клієнта немає можливості відразу передати секретний ключ, зашифрований відкритим ключем сервера.

Сервер створює тимчасову пару ключів *RSA* (відкритий/таємний ключ) і використовує повідомлення *server_key_exchange*, для передавання відкритого ключа клієнтові. Повідомлення включає два параметри тимчасового відкритого ключа *RSA* (значення показника степеня і модуля) та підпис для цих параметрів.

Звичайно підпис створюється за допомогою обчислення хеш-коду повідомлення з подальшим її шифруванням відкритим ключем відправника. У даному випадку хешування визначається формулою

```
hash(ClientHello.random || ServerHello.random || ServerParams).
```

Функція хешування обчислюється для аргументу, який включає не тільки параметри Діффі-Хелмана або *RSA*, але й обидві оказії з оригінальних повідомлень привітання. Це забезпечує захист від атак відтворення повідомлень і атак фальсифікації відповіді. Для підпису *DSS* хеш-код обчислюється за допомогою алгоритму *SHA-1*. Якщо застосовується підпис *RSA*, обчислюється два хеш-коди (за допомогою *MD5* і *SHA-1*), а потім використовується конкатенація отриманих хеш-кодів (36 байт), які шифрується за допомогою відкритого ключа сервера.

Після цього неанонімний сервер (тобто сервер, який використовує не анонімний метод Діффі-Хелмана) може запросити сертифікат клієнта. Повідомлення *certificate_request* (запит сертифіката) включає два параметри: *certificate_type* (тип сертифіката) і *certificate_authorities* (центри сертифікації). Тип сертифіката вказує алгоритм шифрування з відкритим ключем і може бути таким:

- *RSA*, тільки для підпису.
- *DSS*, тільки для підпису.
- *RSA* для методу Діффі-Хелмана з фіксованими параметрами (у цьому випадку підпис забезпечує тільки автентифікацію, яка виконується за допомогою відправлення сертифіката, підписаного за методом *RSA*).
- *DSS* для методу Діффі-Хелмана з фіксованими параметрами (теж забезпечує тільки автентифікацію).
- *RSA* для методу Діффі-Хелмана з одноразовими параметрами.
- *DSS* для методу Діффі-Хелмана з одноразовими параметрами.
- *Fortezza*.

Другим параметром повідомлення *certificate_request* є список імен допустимих центрів сертифікації.

Останнім (і єдиним обов'язковим) повідомленням другого етапу є повідомлення *server_done*, що свідчить про завершення фази вітання сервера. Це повідомлення не має параметрів. Після відправлення цього повідомлення сервер переходить у режим очікування відповіді клієнта.

4.3.8 Автентифікація й обмін ключами клієнта (Етап 3)

Одержавши повідомлення *server_done*, клієнт повинен переконатися в тому, що сервер надав дійсний сертифікат (якщо це потрібно), і що параметри повідомлення *server_hello* є прийнятними. Якщо перевірка завершується успішно, клієнт відправляє серверові одне або декілька повідомлень, мова про які піде нижче.

Якщо сервер запросив сертифікат, клієнт починає даний етап з відправлення серверові повідомлення *certificate*. Якщо в клієнта немає відповідного сертифіката, клієнт відправляє замість нього повідомлення *no_sertificate* (немає сертифіката).

Таким йде повідомлення *client_key_exchange* (обмін ключами клієнта), що для цього етапу є обов'язковим. Вміст цього повідомлення залежить від обраного методу обміну ключами і може бути таким.

RSA. Клієнт генерує 48-байтовий попередній ключ і шифрує його за допомогою відкритого ключа із сертифіката сервера або за допомогою тимчасового ключа *RSA* з повідомлення *server_key_exchange*. Цей попередній ключ дозволяє обчислити головний ключ.

Метод Діффі-Хелмана з одноразовими параметрами або анонімний метод Діффі-Хелмана. Відправляються відкриті параметри клієнта для методу Діффі-Хелмана.

Метод Діффі-Хелмана з фіксованими параметрами. У даному випадку відкриті параметри клієнта для методу Діффі-Хелмана вже були відправлені в повідомленні *certificate*, тому дане повідомлення порожнє.

Fortezza. Відправляються параметри клієнта для алгоритму *Fortezza*.

Для завершення даного етапу клієнт може відправити повідомлення *certificate_verify* (перевірка сертифіката), щоб забезпечити засіб прямої верифікації сертифіката клієнта. Це повідомлення відправляється слідом за сертифікатом клієнта, тобто слідом за будь-яким сертифікатом клієнта, крім тих, які містять параметри Діффі-Хелмана з фіксованими параметрами. Повідомлення включає підпис хеш-коду попереднього повідомлення, обчислений у такий спосіб.

```
Certificate Verify.signature.md5_hash
MD5(master_secret || pad_2 || MD5(handshake_messages || master_secret
|| pad_1)),
Certivicate.signature.sha_hash
SHA(master_secret || pad_2 || SHA(handshake_messages || master_secret
|| pad_1)),
```

де *pad_1* та *pad_2* являють собою значення, визначені вище для *MAC*, *handshake_messages* позначає всі повідомлення протоколу квітерування, відправлені й отримані з моменту відправлення повідомлення *client_hello* (за винятком самого цього повідомлення), а *master_secret* – секретний ключ. Якщо особистий ключ користувача є ключем *DSS*, він використовується для шифрування хеш-коду *SHA-1*. Якщо ж особистий ключ є

ключем *RSA*, то такий ключ використовується для шифрування результату конкатенації хеш-кодів *MD5 і SHA-1*. У будь-якому випадку основна мета – це перевірка наявності у клієнта особистого ключа його сертифіката. Якщо навіть хтось і скористається сертифікатом клієнта, він не зможе відправити таке повідомлення.

4.3.9 Завершення (Етап 4)

Цей етап завершує створення захищеного з'єднання. Клієнт відправляє повідомлення *change_cipher_spec* (зміна параметрів шифрування) і копіює параметри *CipherSpec* стану чекання в поле *CipherSpec* поточного стану. Зверніть увагу на те, що це повідомлення не вважається частиною протоколу квітерування, а відсилається в рамках протоколу зміни параметрів шифрування (*Change Cipher Spec Protocol*). У відповідь клієнт негайно відправляє повідомлення *finished*, шифроване новим алгоритмом з новими ключами і секретними значеннями. Повідомлення *finished* підтверджує, що процеси обміну ключами й автентифікації завершилися успішно. Вміст повідомлення *finished* являє собою конкатенацію таких двох значень хеш-коду.

```
MD5(master_secret || pad_2 || MD5(handshake_messages || Sender ||
master_secret || pad_1)), SHA(master_secret || pad_2 ||
SHA(handshake_messages || Sender || master_secret.|| pad_1)),
```

де *Sender* позначає код, що вказує на те, що відправником є клієнт, а *handshake_messages* включає всі повідомлення квітерування, за винятком даного повідомлення.

У відповідь на ці два повідомлення сервер посилає своє повідомлення *change_cipher_spec*, переводить параметри *CipherSpec* стану очікування в поточний стан і посилає своє повідомлення *finished*. На цьому процес квітерування завершується, і тепер клієнт і сервер можуть почати обмін даними на рівні додатка.

4.4 Захист на рівні протоколів *TLS/SSL*

Основна функція протоколу *TLS* полягає в забезпеченні безпеки та цілісності даних між двома взаємодіючими додатками, один з яких є клієнтом, а інший – сервером.

Протокол *TLS (Transport Layer Security)* розроблявся на основі специфікації протоколу *SSL 3.0 (Secure Socket Layer)*, опублікованого корпорацією *Netscape*. Розходження між даним протоколом й *SSL 3.0* несуттєві, але *TLS 1.0* й *SSL 3.0* несумісні, хоча в *TLS 1.0* передбачений механізм, що дозволяє реалізаціям *TLS* мати сумісність із *SSL 3.0*.

4.4.1 Протокол *TLS*

Перелічимо завдання протоколу *TLS* у порядку їхнього пріоритету.

- **Криптографічна безпека**: *TLS* повинен використовуватися для встановлення безпечного з'єднання між двома учасниками.

- **Інтероперабельність**: незалежні розробники можуть створювати додатки, які будуть взаємодіяти за протоколом *TLS*, що дозволить установлювати безпечні з'єднання.
- **Розширюваність**: *TLS* формує загальний каркас, у який можуть бути вбудовані нові алгоритми відкритого ключа та симетричного шифрування. Це також рятує від необхідності створювати новий протокол, що в свою чергу викличе появу нових слабких місць, і запобігає необхідності повністю реалізовувати нову бібліотеку безпеки.
- **Відносна ефективність**: криптографічні операції інтенсивно використовують центральний процесор, особливо операції з відкритим ключем. Для цього вводиться поняття сесії, для якої визначаються алгоритми та їхні параметри. У рамках однієї сесії може бути створено кілька з'єднань (наприклад, *TCP*). *TLS* дозволяє кешувати сесії для зменшення кількості виконуваних дій при встановленні з'єднання. Це знижує навантаження як на центральний процесор, так і на трафік.

Протокол складається із двох рівнів. Нижнім рівнем, розташованим вище деякого надійного протоколу (а саме, протоколу *TCP*) є протокол *Запису*. Протокол *Запису* забезпечує безпеку з'єднання, яка основана на таких двох властивостях.

- **Конфіденційність з'єднання**. Для захисту даних використовується один з алгоритмів симетричного шифрування. Ключ для цього алгоритму створюється для кожної сесії та базується на секреті, про який домовляються в протоколі *Рукостискання*. Протокол *Запису* також може використовуватися без шифрування.
- **Цілісність з'єднання**. Забезпечується перевірка цілісності повідомлення за **допомогою** *MAC* із ключем. Для обчислення *MAC* використовуються безпечні хеш-функції *SHA-1* й *MD5*. Протокол *Запису* може виконуватися без обчислення *MAC*, але звичайно функціонує в цьому режимі.

Протокол *Запису* використовується для інкапсуляції різних протоколів більш високого рівня. Одним із протоколів більш високого рівня є протокол *Рукостискання*, який використовує протокол *Запису* як транспорт для ведення переговорів про параметри безпеки. Протокол *Рукостискання* дозволяє серверу й клієнтові автентифікувати один одного та домовитися про алгоритми шифрування і криптографічні ключі до того, як прикладний протокол, який виконується на тому ж рівні, почне передавати або приймати перші байти даних.

Протокол *Рукостискання* забезпечує безпеку з'єднання, яка основана на таких властивостях.

- Учасники автентифіковані з використанням криптографії з відкритим ключем (тобто з використанням алгоритмів *RSA*, *DSS* і т.д.). Ця автентифікація може бути необов'язковою, але звичайно потрібна для сервера.

- Переговори про розділений секрет безпечні, тобто цей загальний секрет неможливо підглянути.
- Переговори про розділений секрет надійні, якщо виконана автентифікація хоча б однією зі сторін. У такому випадку атакуючий, розташований у середині з'єднання, не може модифікувати переданий секрет непомітно для учасників з'єднання.

Одна з переваг *TLS* полягає в тому, що він незалежний від прикладного протоколу. Протоколи більш високого рівня можуть прозоро розташовуватися поверх протоколу *TLS*.

В протоколі *TLS* використовується чотири криптографічні операції: цифровий підпис, потокове шифрування, блокове шифрування й шифрування з відкритим ключем.

В операції цифрового підпису вхідними даними для алгоритму підпису є результат застосування однобічної хеш-функції, яка підписує дані. Довжина входу визначається алгоритмом підпису. При використанні алгоритму *RSA* підписується 36-байтова структура, що складається з конкатенації 20 байтів хеш-коду *SHA-1* та 16 байтів хеш-коду *MD5*. При використанні *DSS* 20 байтів хеш-коду *SHA-1* подаються на вхід алгоритму *DSA* без додаткового хешування. При цьому створюється значення *r* та *s*.

При потоковому шифруванні для незашифрованого тексту виконується операція *XOR* з тією ж кількістю значень, створених криптографічно безпечним (із ключем) генератором псевдовипадкових чисел.

При блоковому шифруванні кожен блок незашифрованого тексту шифрується, у результаті чого створюється блок зашифрованого тексту. Всі алгоритми блокового шифрування виконуються в режимі *CBC*, і довжина всіх шифрованих елементів повинна бути кратна довжині блока алгоритму шифрування.

При шифруванні з відкритим ключем використовується алгоритм відкритого ключа, при цьому дані можуть бути дешифровані тільки за допомогою відповідного закритого ключа.

Для одержання *MAC* використовується *HMAC*, тому якщо не знати секрету *MAC*, підробити *MAC* неможливо. *HMAC* може використовуватися з різними хеш-алгоритмами. *TLS* задіє при *Рукостисканні* два алгоритми, *MD5* й *SHA-1*, позначуваних як *HMAC_MD5 (secret, data)* і *HMAC_SHA (secret, data)*. Можуть бути визначені додаткові хеш-алгоритми. В алгоритмі визначена функція, яка розширює секрет до потрібної довжини для створення всіх необхідних ключів. Така псевдовипадкова функція, (*PRF*), одержує як вхід секрет, *seed* (значення, що з однієї сторони є випадковим, а з іншої сторони не є секретним, тобто може стати відоме опонентові) та ідентифікаційну мітку, а також створює вихідне значення необхідної довжини. Для того, щоб зробити псевдовипадкову функцію якомога більш безпечною, використовуються два безпечних хеш-алгоритми. По-перше, визначається функція розширення даних *P_hash*

(secret, data), що застосовує єдину хеш-функцію для розширення секрету й *seed*:

P_hash (secret, seed) = HMAC_hash (secret, A(1) + seed) + + HMAC_hash (secret, A(2) + seed) + HMAC_hash (secret, A(3) + seed) + ...,

де + – позначає конкатенацію.

A() – визначається в такий спосіб: *A(0) = seed, A (i) = HMAC_hash (secret, A (i – 1)).*

P_hash може мати стільки ітерацій, скільки необхідно для створення необхідної кількості даних. Наприклад, якщо *P_SHA-1* використовується для створення 64 байтів даних, то кількість ітерацій повинна дорівнювати 4, при цьому буде створено 80 байтів даних; останні 16 байтів заключної ітерації будуть відкинуті, щоб залишити тільки 64 байти вихідних даних.

Псевдовипадкова функція створюється шляхом розщеплення секрету на дві половини, одна половина використовується для створення даних за допомогою *P_MD5*, а інша – для створення даних за допомогою *P_SHA-1*, після чого обидва виходи цих функцій додаються за модулем 2. Псевдовипадкова функція визначається як результат додавання за модулем 2 цих двох псевдовипадкових значень.

```
PRF(secret,label,seed)=P_MD5(S1,label+seed)⊕P_SHA-1(S2,label+seed);
```

де *Label* – фіксований текстовий рядок.

Оскільки *MD5* створює 16-байтові значення, а *SHA-1* створює 20-байтові значення, то границі внутрішніх ітерацій не будуть збігатися. Наприклад, для створення 80-байтового значення необхідно виконати 5 ітерацій *P_MD5* й 4 ітерації *P_SHA-1*.

Протокол *Запису* складається з декількох рівнів. На кожному рівні повідомлення можуть включати поля довжини, опису та вмісту. Протокол *Запису* фрагментує повідомлення на блоки потрібної довжини, здійснює стискання даних, застосовує *MAC* і зашифровує їх, після чого результат передається по мережі. На іншому кінці з'єднання отримані дані дешифруються, перевіряється їхня цілісність, далі вони декомпресуються, дефрагментуються та передаються протоколам більш високого рівня.

Поверх протоколу *Запису* можуть розташовуватися такі протоколи: протокол *Рукостискання*, *Alert*-протокол, протокол *Зміни параметрів шифрування* й протокол прикладних даних. Для того щоб мати можливість розширення протоколу *TLS*, протокол *Запису* допускає створення нових типів записів. Якщо реалізація *TLS* одержує незрозумілий або непідтримуваний тип запису, то він ігнорується. Тип і довжина запису не зашифровані, отже ці значення не повинні містити секретних даних.

Параметрами виконання протоколу *Запису* є алгоритм стиснення, алгоритм шифрування та *MAC*-алгоритм, а також параметри цих алгоритмів, тобто секрети *MAC*, ключі алгоритму шифрування та вектори ініціалізації. Для кожного напрямку (відповідно читання або запис)

параметри з'єднання можуть розрізнятися. Існує чотири стани з'єднання: поточні та очікувані стани читання/запису. Параметри безпеки для очікуваних станів установлюються протоколом *Рукостискання*.

4.4.2 Обчислення ключів

Протокол *Запису* використовує такий алгоритм для створення ключів, векторів ініціалізації (*IV*) та секретів *MAC* із параметрів безпеки, створюваних протоколом *Рукостискання*.

З майстра-секрету з використанням хеш-функції створюється послідовність байтів, що являє собою *MAC*-секрети, ключі та вектори ініціалізації: *client write MAC secret, server write MAC secret, client write key, server write key, client write IV й server write IV*. Якщо деяке значення не використається, то воно є порожнім.

Для створення ключа обчислюється:

```
key_block = PRF (SecurityParameters.master_secret,
"key expansion", SecurityParameters.server_random +
SecurityParameters.client_random);
```

Обчислення виконуються до тих пір, поки не буде отримано результат заданої довжини. Потім *key_block* розбивається на блоки для одержання необхідних ключів у такий спосіб:

```
client_write_MAC_secret [SecurityParameters.hash_size]
server_write_MAC_secret [SecurityParameters.hash_size]
client_write_key [SecurityParameters.key_material_length]
server_write_key [SecurityParameters.key_material_length]
client_write_IV [SecurityParameters.IV_size]
server_write_IV [SecurityParameters.IV_size]
```

client_write_IV й server_write_IV.

Для експортованих блокових алгоритмів вектора ініціалізації створюються іншим способом. Після виконання даних обчислень вся інформація про секрет та *key_block* скидається. Для експортованих алгоритмів шифрування (для яких *CipherSpec.is_exportable* є *true*) потрібне додаткове обчислення ключів запису:

```
final_client_write_key = PRF (SecurityParameters.client_write_key,
 "client write key", SecurityParameters.client_random +
 SecurityParameters.server_random);
final_server_write_key = PRF (SecurityParameters.server_write_key,
 "server write key", SecurityParameters.client_random +
 SecurityParameters.server_random.
```

Для експортованих алгоритмів шифрування вектора ініціалізації обчислюються в такий спосіб:

```
iv_block = PRF ("", "IV block",  SecurityParameters.client_random +
 SecurityParameters.server_random);
```

IV_block розділяється на два вектори ініціалізації:

```
client_write_IV [SecurityParameters.IV_size]
server_write_IV [SecurityParameters.IV_size].
```

В цьому випадку псевдовипадкова функція використається без секрету, це означає, що секрет має нульову довжину і на результат обчислення псевдовипадкової функції не впливає.

Протокол *Рукостискання* складається із трьох протоколів, використання яких дозволяє учасникам узгодити параметри безпеки для

протоколу *Запису*, автентифікувати один одного, домовитися про параметри безпеки й повідомити один одному про виникнення тих або інших помилок. У результаті виконання протоколу *Рукостискання* будуть створені елементи сесії показані в таблиці 4.3.

Таблиця 4.3 – Створювані елементи сесії

Елемент	Опис
Ідентифікатор сесії	Довільна послідовність байтів, що обирається сервером для ідентифікації активного або поновлюваного стану сесії.
Сертифікат учасника	*X. 509 v3* сертифікат учасника. Цей елемент може бути нульовим.
Метод стиснення	Алгоритм, використовуваний для стиснення даних перед шифруванням.
Набір алгоритмів	Алгоритм симетричного шифрування даних (такий як *null*, *DES* і т.д.), *MAC*-алгоритм (такий як *MD5* або *SHA*) і різні криптографічні атрибути, такі як *hash_size*.
Майстер-секрет	48-байтовий секрет, розділюваний клієнтом і сервером.
Поновлювано	Прапорець, що визначає, чи може дана сесія використовуватися для створення нового з'єднання.

4.4.3 Протокол зміни шифрування

Протокол складається з єдиного повідомлення, яке зашифроване й стиснуте, як визначено в поточному (не очікуваному) стані з'єднання. Повідомлення про зміну шифрування може посилатися як клієнтом, так і сервером для повідомлення одержувача про те, що такі записи будуть захищені алгоритмами та ключами, про які сторони тільки що домовилися. При надходженні даного повідомлення одержувач повинен інформувати протокол *Запису* про негайне копіювання очікуваного стану читання в поточний стан читання.

Відразу після відправки даного повідомлення відправник повинен інформувати протокол *Запису* на своєму кінці з'єднання про негайне копіювання очікуваного стану запису в поточний стан запису. Повідомлення про зміну шифрування посилається при *Рукостисканні* після того, як параметри безпеки погоджені, але перед тим як посилається заключне верифікуюче повідомлення.

4.4.4 Повідомлення закриття

Одним із протоколів, що лежить вище протоколу *Запису*, є протокол *Alert*. Вмістом протоколу є або фатальне, або попереджувальне повідомлення. Фатальне повідомлення повинне приводити до негайного розриву даного з'єднання. У цьому випадку інші з'єднання, що відповідають даній сесії, можуть бути продовжені, але ідентифікатор сесії повинен бути відмічений недійсним для запобігання використанню даної сесії при встановленню нових з'єднань. Подібно іншим повідомленням, повідомлення *Alert* зашифровані й стислі, як визначено в поточному стані з'єднання.

І клієнт, і сервер повинні отримати повідомлення про те, що з'єднання завершується. Кожен учасник може ініціювати обмін повідомленнями закриття. Повідомлення *close_notify* повідомляє одержувача про те, що відправник не буде більше посилати ніяких повідомлень по даному з'єднанні. Сесія стає непоновлюваною, якщо хоча б одне з'єднання завершене без відповідного попереджувального повідомлення *close_notify*. Кожен учасник може ініціювати закриття посилкою повідомлення *Alert* типу *close_notify*. Будь-які дані, відправлені після *Alert*-закриття, ігноруються. Потрібно щоб кожен учасник посилав *close_notify Alert* перед закриттям сторони запису з'єднання. Це означає, що при одержанні відповіді іншого учасника з *Alert* типу *close_notify* з'єднання негайно закривається, а всі очікувані стани скидаються. Ініціаторові закриття не обов'язково чекати відповідного *close_notify Alert*. Якщо прикладний протокол, що використає *TLS*, допускає, що будь-які дані можуть передаватися більш низьким транспортом після того як з'єднання *TLS* закрите, реалізація *TLS* повинна одержувати відповідний *Alert* типу *close_notify* перед тим, як повідомляти прикладному рівню, що з'єднання *TLS* завершене. Якщо прикладний протокол не буде передавати ніяких додаткових даних, а буде тільки закривати більш низьке транспортне з'єднання, реалізація може закрити з'єднання без очікування відповідного *close_notify*. У стандарті *TLS* не визначений спосіб, яким управляються дані транспорту, включаючи відкриття й закриття з'єднань.

4.5 Криптографічні обчислення

Такими питаннями, що викликають інтерес, є створення в ході обміну ключами загального головного секретного ключа і генерування за допомогою цього ключа криптографічних параметрів.

Спільний головний секретний ключ являє собою 48-байтове значення (384 біт), яке використовується один раз та генерується для даного сеансу в ході захищеного обміну ключами. Створення головного ключа складається з двох етапів. На першому етапі узгоджується значення попереднього ключа (*pre_master_secret*), а на другому обидві сторони обчислюють значення головного ключа (*master_secret*). Для передачі значення *pre_master_secret* у сторін є два варіанти.

RSA. Генерований клієнтом 48-байтовий ключ *pre_master_secret* шифрується за допомогою відкритого ключа *RSA* сервера і відправляється клієнтом серверові. Сервер дешифрує отриманий шифрований текст за допомогою свого особистого ключа і відновлює значення *pre_master_secret.*

Метод Діффі-Хелмана. І клієнт, і сервер генерують відкриті ключі для алгоритму Діффі-Хелмана. Після обміну цими ключами кожна сторона виконує певні обчислення за методом Діффі-Хелмана, у результаті яких отримується спільне значення *pre_master_secret.*

Тепер обидві сторони можуть обчислити значення *master_secret* за схемою

```
master_secret = MD5(pre_master_secret || SHA('A' || pre_master_secret ||
ClientHello.random || ServerHello.random)) ||
MD5(pre_master_secret ||SHA('BB' || pre_master_secret ||
ClientHello.random || ServerHello.random)) ||
MD5(pre_master_secret || SHA('CCC || pre_master_secret ||
ClientHello.random || ServerHello.random)),
```

де *ClientHello.random* і *ServerHello.random* є значеннями оказій, які входять в оригінальні повідомлення вітання сторін.

Для поля *CipherSpecs* потрібні секретний ключ *MAC* клієнта для запису, секретний ключ *MAC* сервера для запису, ключ клієнта для запису, ключ сервера для запису, вектор ініціалізації клієнта для запису і вектор ініціалізації сервера для запису. Усі ці параметри генеруються з головного ключа за допомогою застосування функції хешування до головного ключа для одержання захищеної послідовності байтів достатньої довжини.

Процедура генерування ключів з головного ключа аналогічна процедурі генерування головного ключа з попереднього і показана нижче:

```
key_block = MD5(master_secret || SHA('A' || master_secret ||
ServerHello.random || ClientHello.random)) ||
MD5(master_secret || SHA('BB' || master_secret||
ServerHello.random||ClientHello.random)) ||
MD5(master_secret || SHA('CCC' || master_secret ||
ServerHello.random || ClientHello.random)) ||...
```

Процедура виконується до тих пір, поки не буде згенерована послідовність достатньої довжини. Ця алгоритмічна структура являє собою псевдовипадкову функцію. Значення *master_secret* можна розглядати як ініціалізуюче значення для цієї функції. Згенеровані клієнтом і сервером випадкові числа можна розглядати як значення модифікаторів (*salt values*), використовуваних з метою ускладнення криптоаналізу.

4.6 Захист транспортного рівня

Протокол *TLS* є результатом ініціативи *IETF*, метою якої є розробка стандарту *SSL* для *Інтернет*. Поточна версія проекту стандарту *TLS* дуже схожа на *SSLv3*.

Розглянемо розходження між *TLS* і *SSLv3*.

4.6.1 Номер версії

Формат запису *TLS* аналогічний форматові запису *SSL*, і поля заголовка мають той же зміст і призначення. Єдина відмінність полягає в поданні значень, які задають номери версії. Для поточного проекту стандарту *TLS* значенням поля головного номера версії (*Major Version*) є 3, а значенням поля додаткового номера версії (*Minor Version*) – 1.

4.6.2 Код автентичності повідомлення

Схеми обчислення значень *MAC* протоколів *SSL* і *TLS* відрізняються за двома параметрами: алгоритмом, що використовується, та областю даних, для яких обчислюється значення *MAC*. У протоколі *TLS* використовується алгоритм *HMAC*. Алгоритм *HMAC* визначається формулою

$$HMAC_K = H\left[\left(K^+ \oplus opad\right) \| H\left[\left(K^+ \oplus ipad\right) \| M\right]\right],$$

де H – вбудована функція хешування (для *TLS* це *MD5* або *SHA-1*);

M – повідомлення, яке подається на вхід алгоритму *HMAC*;

K^+ – секретний ключ, доповнений зліва нулями, щоб у результаті довжина отриманого значення дорівнювала довжині блока хеш-коду (для *MD5* і *SHA-1* довжина блока дорівнює 512 біт);

opad – значення 00110110 (рівне 36 у шістнадцятковому форматі), повторене 64 рази (512 біт);

ipad – значення 01011100 (рівне 5C в шістнадцятковому форматі), повторене 64 рази (512 біт).

У протоколі *SSL* застосовується такий же алгоритм, але байти заповнювача додаються до секретного ключа за допомогою операції конкатенації, а не пов'язуються з доповненим до потрібної довжини секретним ключем за допомогою операції *XOR* (за винятком "АБО"). Рівень захищеності в обох випадках виявляється однаковим. У *TLS* процес обчислення *MAC* охоплює поля, які використовуються у виразі:

```
HMAC_hash(MAC_write_secret, seq_num || TLSCompressed.type ||
TLSCompressed.version||TLSCompressed.length || TLSCompressed.fragment)).
```

Для обчислення значення *MAC* використовуються всі поля, задіяні при обчисленні *MAC* у протоколі *SSL*, а також поле *TLSCompressed.version*, яке указує версію застосованого протоколу.

4.6.3 Псевдовипадкова функція

У протоколі *TLS* застосовується псевдовипадкова функція (*PRF*) і призначена для розміщення в блоках даних секретних значень, використовуваних при генеруванні ключів або перевірці чи ключі дійсні. Метою її застосування є одержання невеликого по довжині спільного секретного значення для генерування більш довгих блоків даних, захищених від певних типів атак на функції хешування й обчислення значень *MAC*. В основі псевдовипадкової функції лежить така схема розширення даних (рис. 4.11):

```
P_hash(secret, seed) = HMAC_hash(secret, A(1) || seed) ||
 HMAC_hash(secret, A(2) || seed) ||
 HMAC_hash(secret, A(3) || seed) || ...,
```

де *A()* визначається як

```
A(0) = seed,
A(i) = HMAC_hash(secret, A(i -1)),
```

seed позначає ініціалізуюче значення, а *secret* – секретне значення.

Функція розширення даних використовує алгоритм *HMAC* з відповідною функцією хешування (*MD5* або *SHA-1*). Як видно з наведеної вище схеми, функцію *P_hash* можна ітерувати необхідне число раз до одержання блока даних потрібної довжини.

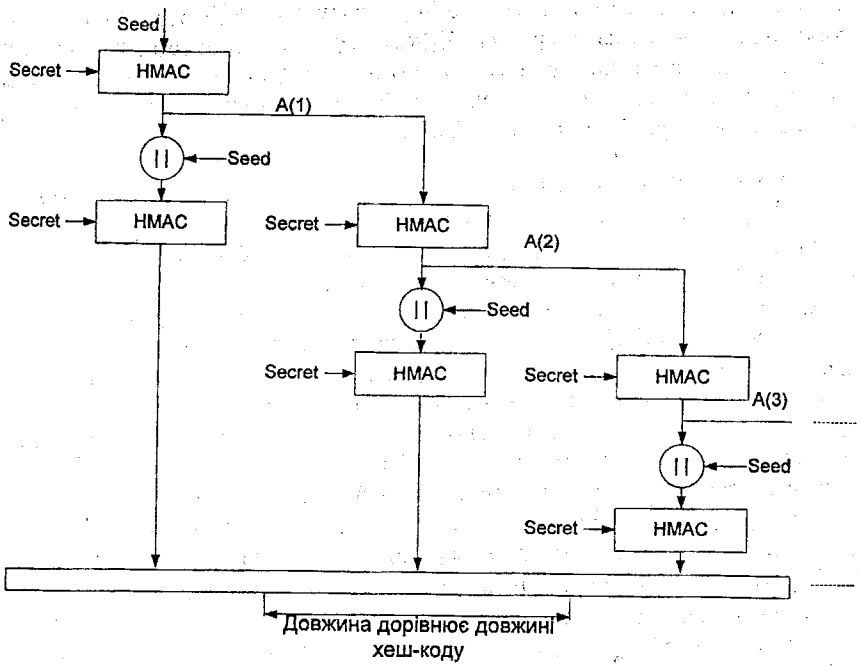


Рисунок 4.11 – Функція P_hash(secret, seed) протоколу TLS

Наприклад, якщо для одержання 64-байтового блока даних застосовується *P_SHA-1*, то для одержання 80 байт даних, з яких 16 останніх байтів будуть відкинуті, цю функцію потрібно буде використовувати повторно 4 рази. У такій же ситуації при використанні *P_MD5* також буде потрібно чотири ітерації, але в результаті буде отримано рівно 64 байт даних. Зверніть увагу на те, що на кожній ітерації алгоритм *HMAC* виконується двічі і щораз двічі виконується алгоритм хешування, який є його основою.

Для того, щоб зробити псевдовипадкову функцію максимально захищеною, в ній передбачено два різних алгоритми хешування, що гарантує захищеність даної функції, поки принаймні один з цих алгоритмів залишається захищеним. Псевдовипадкова функція *PRF* визначена в такий спосіб.

$$PRF(secret, label, seed) = P\_MD5(S1, label \,||\, seed) \otimes P\_SHA\text{-}1(S2, label \,||\, seed).$$

На вхід псевдовипадкової функції надходить секретне значення, ідентифікуюча мітка та ініціалізуюче значення, а на вихід – блок даних будь-якої підходящої довжини. Вихідне значення отримано шляхом поділу секретного значення на дві частини (*S1* і *S2*) і застосування до кожної з цих частин функції *P_hash,* причому при обробці однієї частини

використовується алгоритм *MD5*, а іншої частини – *SHA-1*. Два отриманих значення пов'язуються операцією *XOR*, у результаті чого отримано вихідне значення (для того, щоб в результаті зв'язування операцією *XOR* вийшли блоки даних однакової довжини, для *P_MD5* потрібно використовувати більше число ітерацій, ніж для *P_SHA-1*).

4.6.4 Коди попереджень

Протокол *TLS* підтримує всі типи повідомлень, визначені для *SSL*, за винятком повідомлення *no_sertificate* (немає сертифіката). Але для *TLS* визначений цілий ряд додаткових кодів повідомлення. Такі додаткові коди завжди означають критичну помилку.

- *decription_failed* – шифрований текст дешифрований неправильно: його довжина не кратна заданій довжині блока або перевірка показала наявність некоректного значення заповнювача;
- *record_overflow* – отримано запис *TLS* з корисним вантажем (шифрованим текстом), довжина якого перевищує 214 + 2048 байт, або після дешифрування довжина тексту стала більша 214 + 1024 байт;
- *unknown_ca* – отримано достовірний сертифікат або його частина, але сертифікат не був визнаний, тому що відповідний центр сертифікації не був знайдений, або сертифікат неможливо поставити у відповідність з відомим надійним центром сертифікації;
- *access_denied* – отримано дійсний сертифікат, але в процесі застосування засобів керування доступом відправник прийняв рішення не продовжувати переговори;
- *decode_error* – повідомлення не може бути декодовано, тому що одне з полів містить значення, яке виходить за рамки допустимого діапазону, або довжина повідомлення виявилася неправильною;
- *export_restriction* – у процесі переговорів було виявлено, що порушуються експортні обмеження на довжину ключа;
- *protocol_version* – номер версії протоколу, запропонованого клієнтом у процесі переговорів, розпізнаний, але не підтримується;
- *insufficient_security* – повертається замість *handshake_failure* (аварійне завершення процесу квітерування) у тих випадках, коли переговори завершуються аварійно, оскільки сервер вимагає більш захищених шифрів, ніж ті, які підтримуються клієнтом;
- *internal_error* – виникла не пов'язана з другою стороною або коректністю роботи протоколу внутрішня помилка, що не дозволяє продовжити процес;
- *decrypt_error* – аварійне завершення деякої криптографічної операції, виконуваної в ході переговорів, наприклад, операції перевірки підпису, дешифрування даних обміну ключами або перевірки дійсності повідомлення *finished*;
- *user_cancelled* – процес завершений з деякої причини, не пов'язаної з

роботою протоколу;

- *no_renegotiation* – відсилається клієнтом у відповідь на запит *hello* (вітання) або сервером у відповідь на запит *hello* клієнта після ініціалізації процесу переговорів. У звичайній ситуації такий запит приводить до поновлення переговорів, але одержання даного повідомлення вказує на те, що його відправник не може відновити переговори. Дане повідомлення завжди є попередженням.

4.6.5 Пакети шифрів

Протоколи *SSL* і *TLS* мають такі невеликі розходження у використовуваних ними комплектах шифрів.

Обмін ключами. *TLS* підтримує всі методи обміну ключами, реалізовані в *SSL*, за винятком *Fortezza*.

Алгоритми симетричної схеми шифрування. У *TLS* включені всі алгоритми симетричної схеми шифрування, за винятком *Fortezza*.

4.6.6 Типи сертифікатів клієнта

Для *TLS* визначені такі типи сертифікатів, запитуваних у повідомленні *certificate_request* (запит сертифіката): *rsa_sign, dss_sign, rsa_fixed_dh* та *dss_fixed_dh*. Усі ці типи визначені і для *SSL*. Крім того, протокол *SSL* підтримує типи *rsa_ephemeral_dh, dss_ephemeral_dh* та *fortezza_kea*. В алгоритмі Діффі-Хелмана з одноразовими параметрами передбачається підпис параметрів Діффі-Хелмана за допомогою *RSA* або *DSS*. У протоколі *TLS* цієї функції відповідають типи *rsa_sign* і *dss_sign*, тому вводити окремий тип для позначення підпису параметрів Діффі-Хелмана не потрібно. Схема *Fortezza* протоколом TLS не підтримується.

У повідомленні *certificate_verify* (перевірка сертифіката) протоколу *TLS* хеш-коди *MD5* і *SHA-1* обчислюються тільки для повідомлень *handshake_messages*. Нагадаємо, що в *SSL* при обчисленні хеш-коду використовуються також головний ключ і заповнювачі. Схоже, що додавання цих значень не збільшує надійність захисту.

Що стосується повідомлення *finished*, то в *TLS* воно являє собою хеш-код, обчислений за допомогою спільного значення *master_secret*, попередніх повідомлень квітерування і мітки, яка ідентифікує клієнта або сервера. Обчислення відбуваються за такою, відмінною від *SSL*, схемою:

```
PRF(master_secret, finished_label, MD5(handshake_messages) || SHA-1
(handshake_messages)),
```

де *finished_label* позначає рядок "*client finished*" для клієнта або "*server finished*" для сервера.

4.6.7 Криптографічні обчислення

Значення *pre_master_secret* (попередній ключ) для *TLS* обчислюється так само, як і для *SSL*. Як і в *SSL*, значення *master_secret* (головний ключ) для *TLS* теж обчислюється як хеш-код значення *pre_master_secret* і двох

випадкових чисел з повідомлень *hello*. Однак для *TLS* обчислення виконуються за іншою схемою:

```
master_secret=PRF(pre_master_secret,"master_secret",ClientHello.rando
m||ServerHello.random).
```

Алгоритм виконується доти, поки на виході не буде отримана псевдовипадкова послідовність довжиною в 48 байт.

Обчислення блока ключів (секретні ключі *MAC*, сеансові ключі шифрування і вектори ініціалізації) виконується за схемою

```
key_block =PRF(master_secret, "key expansion",
SecurityParameters.server_random|| SecurityParameters.client_random),
```

обчислення продовжуються доти, поки не буде згенерована послідовність потрібної довжини. Як і в *SSL*, значення *key_block* являє собою функцію значення *master_secret* і випадкових значень, згенерованих клієнтом і сервером, але алгоритм для обчислення у *TLS* є іншим.

4.6.8 Заповнення

У *SSL* байти заповнювача додаються до даних користувача, які підлягають шифруванню, у мінімально необхідній кількості, достатній для того, щоб одержати загальну довжину даних для шифрування, кратну довжині блока шифру. У *TLS* дозволяється додавати будь-яке число заповнювачів (до 255 байт включно), за умови, що в результаті довжина блока даних кратна довжині блока шифру. Наприклад, якщо відкритий текст разом зі значенням *MAC* і байтом *padding.length* складають 79 байт, то для заповнення може використовуватися 1, 9, 17, ..., 249 байт. Змінювану довжину заповнення можна використовувати для того, щоб утруднити спроби аналізу довжин переданих повідомлень.

5 ЗАХИСТ МЕРЕЖЕВОГО КЕРУВАННЯ

Мережі і розподілені системи обробки даних відіграють винятково важливу роль у сфері бізнесу, в урядових та інших організаціях. У рамках окремих організацій спостерігається тенденція розширення й ускладнення мереж, які підтримують усе більше число додатків і обслуговують усе більше число користувачів. Зі збільшенням мереж стають очевидними два такі факти:

- мережа та ресурси, які вона поєднує, разом з розподіленими програмними додатками стають украй необхідними для організації;
- виникає усе більше складностей і несправностей, які призводять до відмов частин мережі або уповільнення швидкості роботи мережі до неприйнятного рівня.

Велика мережа просто не може будуватися та керуватися вручну. Складність системи змушує використовувати автоматизовані засоби мережевого керування. Якщо в мережі використовується устаткування різних виробників, необхідність застосування таких засобів стає ще більш гострою. У зв'язку з цим були розроблені стандарти мережевого керування, які стосуються сервісів, протоколів і інформаційних баз даних системи керування. Безумовно, найбільше широко використовуваним стандартом такого типу є *SNMP (Simple Network Management Protocol* – простий протокол мережевого керування).

З часу його публікації в 1988 році, протокол *SNMP* знаходив своє застосування у все більшому числі мереж і в усе більш складних середовищах. З поширенням *SNMP* виникала необхідність у нових функціональних можливостях, що відбивають нові вимоги. Зокрема, стала очевидною важливість забезпечення захисту як невід'ємної частини мережевого керування. З метою розширення функціональних можливостей була розроблена версія 2 протоколу *SNMP*. Удосконалені функції захисту було опубліковано в *SNMPv3*.

5.1 Основні принципи роботи протоколу *SNMP*

5.1.1 Архітектура мережевого керування

Система мережевого керування – це сукупність інструментальних засобів мережевого моніторингу та керування, інтегрованих у єдине ціле, і вирішує такі задачі:

- забезпечення єдиного робочого інтерфейсу із широкими можливостями та набором команд, які дозволяють виконувати більшість або всі задачі мережевого керування;
- використання вже наявного устаткування шляхом впровадження в нього додаткових апаратних засобів і програмного забезпечення керування мережею.

Система мережевого керування складається з додаткових апаратних засобів і програмного забезпечення, які використовуються разом з наявними мережевими компонентами. Програмне забезпечення, яке виконує функції мережевого керування, розміщається в хостах і процесорах зв'язку (наприклад, комунікаційних процесорах, термінальних групових контролерах). Система мережевого керування повинна відображати всю архітектуру мережі з адресами та мітками, призначеними кожній точці мережі, і конкретними атрибутами всіх елементів та з'єднань, відомих системі. Регулярне надходження інформації про стан мережі в центр мережевого керування забезпечується активними елементами мережі.

Модель мережевого керування, яка використовується у протоколі *SNMP*, складається з таких ключових елементів:

- станція керування;
- агент керування;
- інформаційна база системи керування;
- протокол мережевого керування.

Станція керування звичайно являє собою автономний пристрій, але може бути і набором можливостей, реалізованих у загальнодоступній системі. У будь-якому випадку станція керування виконує роль інтерфейсу між людиною в особі мережевого адміністратора і системою мережевого керування. Станція керування повинна мати, як мінімум, такі можливості:

- набір керуючих додатків для аналізу даних, відновлення після збоїв тощо;
- інтерфейс, за допомогою якого мережевий адміністратор може здійснювати моніторинг мережі і керування мережею;
- можливість перекладу вимог мережевого адміністратора в реальні дії моніторингу і керування у віддалених точках мережі;
- наявність бази даних з інформацією, витягнутою з інформаційних баз систем керування всіх керованих об'єктів мережі;

Об'єктами стандартизації *SNMP* є тільки два останніх елементи цього списку.

Іншим активним елементом системи мережевого керування є агент керування. Ключові елементи платформи, такі як хости, мости, маршрутизатори і концентратори, можуть підтримувати *SNMP*, щоб ними можна було керувати зі станції керування. Агент керування відповідає на запити інформації, що виходять з боку станції керування, реагує на запити дії станції керування і може асинхронно забезпечити станцію керування важливою інформацією без відповідного запиту.

Для керування ресурсами в мережі кожен ресурс вважається об'єктом. Об'єкт, по суті, є змінною, яка являє собою один з аспектів керованого агента. Сукупність об'єктів називається інформаційною базою системи керування (*Management Information Base – MIB*). База *MIB* розглядається

станцією керування як сукупність точок доступу до агента. Об'єкти цієї бази даних стандартні для всіх систем одного класу (наприклад, усі мости підтримують одні і ті самі об'єкти керування). Станція керування здійснює моніторинг, витягаючи відповідні значення об'єктів *MIB*. Станція керування може ініціювати виконання певних дій агентом або ж змінити конфігурацію агента шляхом зміни значень конкретних об'єктів.

Станція керування та агенти пов'язуються протоколом мережевого керування. Протокол, використовуваний для керування мережами *TCP/IP*, називається *SNMP*. Цей протокол реалізовує такі основні можливості.

- *Get* (отримування). Дозволяє станції керування з'ясовувати значення об'єктів в агенті.
- *Set* (установлення). Дозволяє станції керування встановлювати значення об'єктів в агенті.
- *Notify* (повідомлення). Дозволяє агентові сповістити станцію керування про важливі події.

5.1.2 Архітектура протоколу мережевого керування

Структури протоколу і *MIB* досить прості, що дозволяє відносно просто досягти узгодженості дій між станціями керування і програмним забезпеченням агентів різних виробників.

Трьома основними специфікаціями є такі.

- Структура та ідентифікація інформації системи керування для мереж на базі *TCP/IP*. Описує спосіб визначення керованих об'єктів *MIB*.
- Інформаційна база системи керування для міжмережевого середовища на базі *TCP/IP: MIB-II*. Визначає керовані об'єкти *MIB*.
- Простий протокол мережевого керування. Визначає протокол, який використовується для керування цими об'єктами.

SNMP є протоколом прикладного рівня в складі стеку протоколів *TCP/IP*. Він призначений для роботи з протоколом *UDP*. Для автономної станції керування процес адміністратора керує доступом до центральної структури *MIB* у системі станції керування і забезпечує мережевому адміністраторові інтерфейс доступу. Процес адміністратора дає можливість керувати мережею за допомогою реалізації протоколу *SNMP* над протоколами *UDP, IP* і відповідними спеціальними мережевими протоколами (наприклад, *Ethernet, FDDI, X.25)*.

Кожен агент теж повинен підтримувати протоколи *SNMP, UDP* та *IP*. Крім того, є процес агента, який інтерпретує повідомлення протоколів *SNMP* та агента, який керує структурою *MIB*. Для агента, який підтримує інші протоколи (наприклад *FTP)*, поряд з протоколом *UDP* може знадобитися підтримка *TCP*.

На рис. 5.1 показаний вміст протоколу *SNMP*. У системі станції керування від додатка керування виходить три типи повідомлень *SNMP: GetRequest, GetNextRequest* та *SetRequest*. Перші два повідомлення є

варіантами функції *Get*. Усі три повідомлення підтверджуються агентом у формі повідомлення *GetResponse*, яке передає додаткові керування. Крім того, агент може генерувати повідомлення переривання у відповідь на подію, що змінює *MIB* і впливає на відповідні керовані ресурси.

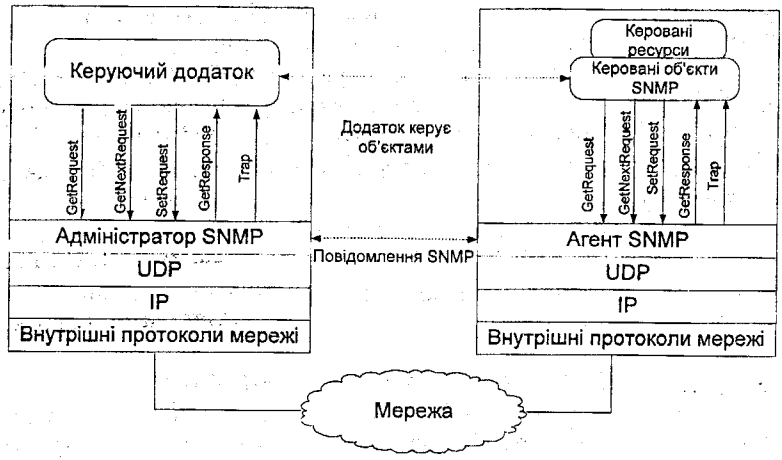


Рисунок 5.1 – Вміст протоколу SNMP

Оскільки протокол *SNMP* опирається на протокол *UDP*, який не вимагає встановлення з'єднання, *SNMP* теж не вимагає з'єднань. Кожен обмін повідомленнями між станцією керування й агентом розглядається як окрема транзакція.

5.1.3 Модулі-посередники

У протоколі *SNMPv1* всі агенти (так само, як і станції керування) повинні підтримувати протоколи *UDP* і *IP*. Це обмежує можливості прямого керування тільки такими пристроями і не дозволяє керувати іншими (наприклад, деякими мостами і модемами, які не підтримують набір протоколів *TCP/IP*). Крім того, у мережі може існувати багато малих систем (персональних комп'ютерів, робочих станцій, програмувальних контролерів), які використовують *TCP/IP* для підтримки своїх додатків, але для яких виявляється недоцільним нести додатковий вантаж підтримки *SNMP*.

Щоб використовувати пристрої, які не підтримують протокол *SNMP*, було запропоновано задіяти механізм посередництва (від англійського *proxy* – доручення, заступник, уповноважений). У відповідній схемі як уповноважений агент одного або відразу декількох пристроїв виступає деякий агент *SNMP*, тобто агент *SNMP* діє як посередник.

На рис. 5.2 показана схема архітектури протоколу, яка використовується найчастіше.

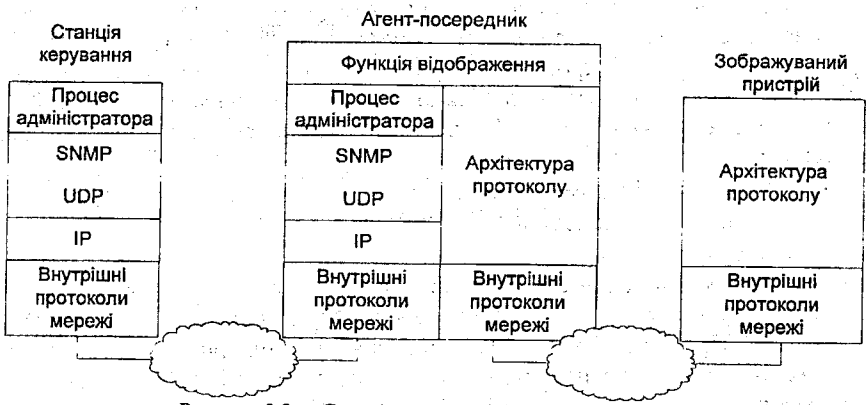


Рисунок 5.2 – Схема архітектури протоколу SNMP

Станція керування посилає запити, які стосуються стану деякого пристрою, уповноваженому агентові-посередникові. Агент-посередник конвертує кожен запит у повідомлення, яке відповідає протоколові керування, що використовується цим пристроєм. Одержавши відповідь на запит, агент передає цю відповідь станції керування. Так само при одержанні агентом від відповідного пристрою повідомлення про деяку подію, агент-посередник посилає станції керування відповідне повідомлення у вигляді переривання.

5.1.4 Конфігурація модуля посередника (система *proxy*)

Протокол *SNMPv2* допускає використання не тільки стеку протоколів *TCP/IP*. Зокрема, *SNMPv2* може використовуватися в рамках набору протоколів *OSI (Open System Interconnection* – взаємодія відкритих систем). Тому *SNMPv2* застосовується для керування різними мережами. В цьому випадку, будь-який пристрій, що не підтримує *SNMPv2*, може керуватися тільки засобами *proxy*, це стосується і пристроїв *SNMPv1*. Таким чином, якщо в пристрої реалізоване програмне забезпечення агента *SNMPv1*, доступ до нього може бути отриманий з боку адміністратора *SNMPv2* тільки через пристрій-посередник, у якому реалізовані агент *SNMPv2* і програмне забезпечення адміністратора *SNMPv1*.

Крім того, *SNMPv2* підтримує внутрішні зв'язки посередництва (*native proxy relationship*), при яких пристрій підтримує *SNMPv2*. У цьому випадку адміністратор *SNMPv2* звертається до вузла *SNMPv2*, який виступає як агент. Такий вузол потім виступає як адміністратор при доступі до пристрою, який тепер відіграє роль агента *SNMPv2*. Причина підтримки такого непрямого звертання до пристроїв полягає в необхідності дати користувачам можливість будувати ієрархічні децентралізовані системи мережевого керування.

Протокол *SNMP* пропонує базовий набір засобів мережевого керування в рамках пакета, які легко реалізувати і налаштувати. Однак з ростом популярності протоколу *SNMP* у сфері керування мережами, які постійно розширюються з постійно зростаючим навантаженням, його недоліки стали очевидними. Ці недоліки можна розділити на такі три категорії:

- відсутність підтримки розподіленого мережевого керування;
- функціональні недоліки;
- недоліки захисту.

Недоліки, які відносяться до двох перших категорій, були виправлені в *SNMPv2*, опис якого було опубліковано в 1993 році, а виправлена версія – у 1996 р. Протокол *SNMPv2* швидко знайшов підтримку, і цілий ряд постачальників оголосили про вихід продуктів, які підтримують цей протокол, усього через кілька місяців після публікації стандарту. Недоліки захисту були виправлені в *SNMPv3*.

5.1.5 Розподілене мережеве керування

У традиційній централізованій схемі мережевого керування один головний комп'ютер відіграє роль станції мережевого керування (можливе існування ще декількох станцій керування, що виконують роль резервних). Інші пристрої мережі містять програмне забезпечення агента й інформаційної бази системи керування *(MIB)*, що дозволяє здійснювати моніторинг стану пристроїв і керування ними зі станції керування. З розширенням мережі та збільшенням інтенсивності трафіка така централізована система зрештою стає зовсім непридатною для використання на практиці. Занадто велике навантаження припадає на станцію керування і занадто інтенсивний потік даних утворюють повідомлення із запитами і відповідями, що йдуть від кожного окремого агента через усю мережу до центра керування. За таких умов децентралізований розподілений підхід виявляється кращим рішенням (наприклад, рис. 5.3).

Децентралізована схема мережевого керування допускає наявність множини станцій керування верхнього рівня, який можна назвати серверами керування. Кожен такий сервер може здійснювати безпосереднє керування деякою частиною загального набору агентів. Але для більшості цих агентів сервер керування делегує відповідальність керування деякому проміжному адміністраторові. Проміжний адміністратор бере на себе відповідальність за здійснення моніторингу і керування агентами. Але він також відіграє роль агента, забезпечуючи інформацією сервер керування верхнього рівня і підкоряючись його керуванню.

Цей тип архітектури розподіляє навантаження більш рівномірно і знижує загальну інтенсивність мережевого трафіка.

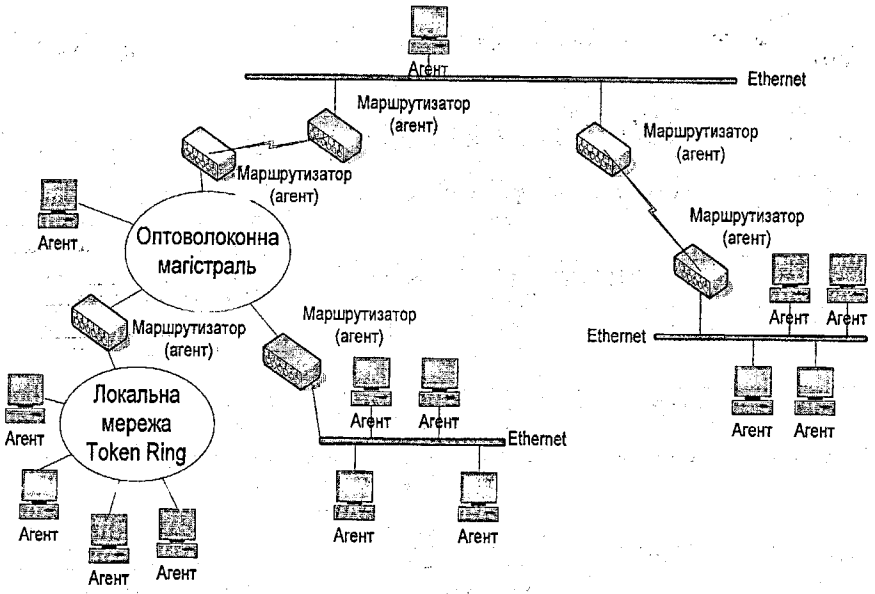


Рисунок 5.3 – Децентралізована схема керування

5.1.6 Приклад конфігурації системи розподіленого мережевого керування

SNMPv2 підтримує або строго централізовану, або розподілену стратегію мережевого керування. В останньому випадку деякі системи виступають як у ролі адміністратора, так і в ролі агента. У ролі агента така система приймає команди вищої системи керування. При цьому одні команди мають відношення до локальної структури *MIB* у системі агента, а інші команди потребують, щоб агент виступав як посередник при звертанні до віддалених пристроїв. Агент-посередник спочатку відіграє роль адміністратора для доступу до інформації віддаленого агента, а потім – роль агента при передачі цієї інформації адміністраторові вищого рівня.

5.1.7 Функціональні вдосконалення

У табл. 5.1 пропонується список функціональних удосконалень, пропонованих *SNMPv2*.

Обидві версії протоколу визначаються в термінах наборів команд, які формують при обміні протокольні одиниці даних *(Protocol Data Unit – PDU)*. У *SNMPv1* таких команд п'ять. Команда *Get* виходить від адміністратора до агента з метою одержання значень об'єктів *MIB*.

Таблиця 5.1 – Порівняння протокольних одиниць обміну *SNMPv1* і *SNMPv2*

SNMPv1 PDU	*SNMPv2 PDU*	Напрямок	Опис
GetRequest	*GetRequest*	Від адміністратора до агента	Запит значення для кожного із зазначених об'єктів
GetNextRequest	*GetNextRequest*	Від адміністратора до агента	Запит такого значення для кожного із зазначених об'єктів
–	*GetBulkRequest*	Від адміністратора до агента	Запит множини значень
SetRequest	*SetRequest*	Від адміністратора до агента	Установлення значення для кожного із зазначених об'єктів
–	*InformRequest*	Від адміністратора до адміністратора	Передача інформації без запиту
GetResponse	*Response*	Від агента до адміністратора або від адміністратора до адміністратора (*SNMPv2*)	Відповідь на запит адміністратора
Trap	*SNMPv2-Trap*	Від агента до адміністратора	Передача інформації без запиту

Команда *GetNext* використовує те, що об'єкти *MIB* утворюють деревоподібну структуру. Коли деякий об'єкт вказується в команді *GetNext*, агент знаходить у дереві такий об'єкт і повертає відповідне значення. Команда *GetNext* виявляється корисною оскільки дає можливість адміністраторові "бродити" по дереву агента, коли не відомий точний набір об'єктів, підтримуваних відповідним агентом. Команда *Set* дозволяє адміністраторові змінювати значення в системі агента, а також створювати і видаляти рядки в таблицях. Команда *GetResponse* використовується агентом для реагування на команди адміністратора. Нарешті, команда *Trap* дає агентові можливість послати інформацію адміністраторові, не чекаючи керуючого запиту. Наприклад, агент може бути побудований так, щоб повідомлення *Trap* посилалося при аварійному завершенні зв'язку або тоді, коли інтенсивність трафіка перевищує заданий поріг.

SNMPv2 включає всі команди, які використовуються в *SNMPv1*, і дві нові. Найбільш важливою з них є команда *Inform*. Ця команда посилається однією станцією керування іншій і, подібно перериванню, несе інформацію, яка стосується умов або подій у системі відправника. Перевага команди *Inform* полягає в тому, що її можна використовувати для створення конфігурації, у рамках якої існує можливість взаємодії множини адміністраторів, що розділяють загальні обов'язки керування, у великій мережі.

Друга нова команда, *GetBulk*, дозволяє адміністраторові відразу одержати великий блок даних. Зокрема, з її допомогою можна пересилати

заповнені даними таблиці.

І ще одна відмінність: команда *Get* є атомарною в *SNMPv1*, але не *SNMPv2*. Якщо команда *Get SNMPv1* містить список об'єктів, для яких запитуються значення, і виявляється, що принаймні один з цих об'єктів у системі агента не існує, то буде відкинута вся команда. У *SNMPv2* можуть повертатися часткові результати. Не є атомарною і команда *Get*, що забезпечує можливість більш ефективного використання мережевих ресурсів адміністратором.

5.2 Об'єднання *SNMPv1* як засоби захисту

Протокол *SNMPv1* пропонує лише елементарні засоби захисту, основані на використанні об'єднань.

5.2.1 Об'єднання та імена об'єднань

Подібно іншим розподіленим додаткам, мережеве керування припускає взаємодію ряду прикладних компонентів, які підтримують визначений протокол. У мережевому керуванні *SNMP* прикладними компонентами є додатки адміністратора та агента, які використовують протокол *SNMP*.

Мережеве керування *SNMP* має деякі особливості в порівнянні з іншими розподіленими додатками. Даний додаток припускає наявність відносини типу "один-множина" між адміністратором і множиною агентів: адміністратор може одержувати і встановлювати значення об'єктів агентів, а також одержувати переривання від агентів. Тому, з погляду роботоздатності і керованості системи, адміністратор "керує" множиною агентів. Може існувати кілька адміністраторів, які керують всіма агентами або деякою підмножиною всіх агентів. Підмножини можуть при цьому перетинатися.

Необхідно також мати можливість зображати мережеве керування *SNMP* як відношення типу "один-множина" між одним агентом і множиною адміністраторів. Кожен агент керує своєю власною локальною базою *MIB* і повинен контролювати її використання адміністраторами. Таке керування складається з таких компонентів.

- **Сервіс автентифікації**. Агент може обмежити доступ до *MIB*, дозволивши його тільки авторизованим адміністраторам.
- **Політика доступу**. Агент може визначити різні привілеї доступу різним адміністраторам.
- **Сервіс посередництва** *(proxy)*. Агент може виступати як посередник перед іншими агентами. Це може означати використання сервісу автентифікації і/або політики доступу для інших агентів у системі посередника.

Усі ці складові керування стосуються захисту. Якщо відповідальність за мережеві компоненти розподіляється між декількома адміністративними об'єктами, то агентам приходиться захищати себе і свої бази *MIB* від небажаного або несанкціонованого доступу. Протокол *SNMP* при цьому пропонує тільки прості можливості захисту – використання об'єднань.

Об'єднання *SNMP* є відношенням між агентом *SNMP* і множиною адміністраторів *SNMP*, які визначають можливості автентифікації і керування доступом, а також характеристики посередництва. Об'єднання – це локальне поняття, визначене в системі агента. Агент установлює по одному об'єднанню для кожної необхідної комбінації автентифікації, керування доступом і характеристик посередництва. Кожне об'єднання одержує унікальне (у рамках цього агента) ім'я, і адміністратори в рамках цього об'єднання повинні використовувати відповідне ім'я об'єднання при виконанні всіх операцій *get* і *set*. Агент може установити кілька об'єднань, причому кожен адміністратор не обов'язково повинен входити тільки в одне з таких об'єднань.

Оскільки об'єднання визначаються агентом локально, одне й те саме ім'я може використовуватися різними агентами. Збіг імен при цьому не має ніякого значення і не вказує на подібність між конкретними об'єднаннями. Тому адміністратор повинен стежити за ім'ям або іменами об'єднань для всіх агентів, до яких адміністраторові необхідно мати доступ.

5.2.2 Сервіс автентифікації

Мета служби автентифікації *SNMPv1* – надати одержувачеві гарантії того, що повідомлення *SNMPv1* виходить від зазначеного в повідомленні джерела. *SNMPv1* передбачає тільки тривіальну схему автентифікації. Кожне повідомлення (запит *get* або *set)*, що направляється від адміністратора до агента, включає ім'я об'єднання. Це ім'я служить паролем, і повідомлення вважається справжнім, якщо відправник знає цей пароль.

При такій обмеженій формі автентифікації багато мережевих адміністраторів неохоче згоджуються на використання засобів, відмінних від моніторингу, для якого потрібні тільки операції *get* і *trap*. Мережеве керування з використанням операції *set,* дуже вразливо з погляду безпеки. Ім'я об'єднання може використовуватися для активізації процедури автентифікації, і тоді воно виконує функції пароля першої лінії захисту. Процедура автентифікації може припускати застосування шифрування/дешифрування, для забезпечення більш надійного захисту.

5.2.3 Політика доступу

За допомогою визначення об'єднання агент дозволяє доступ до своєї структури *MIB* тільки обраній множині адміністраторів. Використовуючи кілька об'єднань, агент може забезпечити різним адміністраторам різні права доступу до *MIB*. Таке керування доступом характеризується такими двома аспектами.

- Подання *SNMP MIB* – підмножина об'єктів усередині *MIB*. Для кожного об'єднання можуть визначатися свої подання *MIB*. Набір об'єктів у поданні не зобов'язаний належати одному і тому ж піддереву *MIB*.

- Режим доступу *SNMP* – елемент із множини *{READ-ONLY, READ-WRITE}* (тільки для читання, читання і запис). Режим доступу визначається для кожного об'єднання.

Комбінація подання *MIB* і режиму доступу називається **профілем об'єднання** *SNMP*. Таким чином, профіль об'єднання складається з певної підмножини *MIB* у системі агента і режиму доступу до відповідних об'єктів. Режим доступу *SNMP* застосовується одночасно до всіх об'єктів даного подання *MIB*. Так, якщо обрано режим доступу *READONLY*, то це відноситься до всіх об'єктів подання та обмежує доступ адміністраторів у рамках цього подання тільки операціями читання.

Профіль об'єднання одержує кожне об'єднання, визначене агентом. Комбінація об'єднання *SNMP* і профілю об'єднання *SNMP* називається політикою доступу *SNMP*. На рис. 5.4 наводиться ілюстрація концепцій, які щойно обговорювалися.

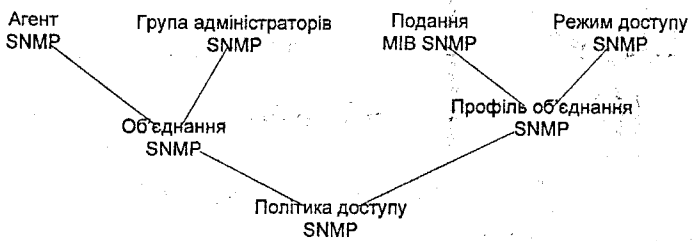


Рисунок 5.4 – Принципи адміністрування SNMPv1

5.2.4 Сервіс посередництва

Поняття об'єднання виявляється корисним і для підтримки сервісу посередництва. Нагадаємо, що посередником є агент *SNMP*, який виступає від імені інших пристроїв. Звичайно інші пристрої є сторонніми, тобто не підтримують *TCP/IP* і *SNMP*. У деяких випадках пристрій може підтримувати *SNMP*, і тоді посередництво використовується для того, щоб звести до мінімуму взаємодію між пристроєм і системами мережевого керування.

Для кожного пристрою, що подається системою-посередником, ця система реалізує відповідну політику доступу *SNMP*. Таким чином, система-посередник знає, які об'єкти *MIB* можуть використовуватися для керування пристроєм (подання *MIB*), і який режим доступу до них.

5.3 *SNMPv3*

У 1998 році робоча група *IETF* зі створення *SNMPv3* опублікувала набір стандартів *Інтернет*.

На рис. 5.5 за допомогою відповідних форматів показаний зв'язок між різними версіями *SNMP*. Обмін інформацією між станцією керування

та агентом відбувається у формі повідомлень *SNMP*. Пов'язана із захистом обробка даних відбувається на рівні повідомлення. Так, наприклад, *SNMPv3* визначає модель *USM (User Security Model* – користувацька модель захисту), що припускає використання полів заголовка повідомлення. Корисним вантажем повідомлення *SNMP* є протокольна одиниця обміну *(PDU) SNMPv1 або SNMPv2. PDU* указує тип дії (наприклад, одержання або установлення значень керованого об'єкта) і список змінних, пов'язаних з цією дією.

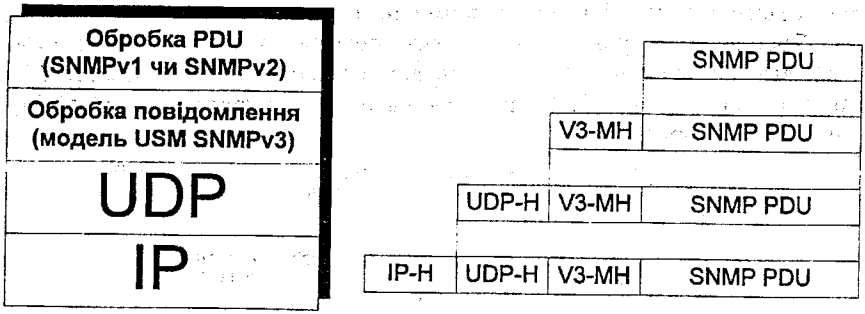


Рисунок 5.5 – Зв'язок між версіями SNMP

IP-H – заголовок *IP;*
UDP-H – заголовок *UDP;*
V3-MH – заголовок повідомлення *SNMPv3.*

5.4 Архітектура протоколу *SNMPv3*

Таким чином, у рамках нової архітектури може використовуватися існуючий формат протокольної одиниці обміну *SNMPv1* або *SNMPv2.* Реалізація протоколу, названа *SNMPv3*, складається із засобів захисту й особливостей архітектури, а також формату *PDU.*

5.4.1 Архітектура *SNMP*

Архітектура *SNMP* складається з розподіленої сукупності взаємодіючих об'єктів *SNMP.* Кожен об'єкт реалізовує деяку порцію можливостей *SNMP* і може виступати як вузол-агент, вузол-адміністратор або деяка їхня комбінація. Всі об'єкти *SNMP* складаються із сукупності модулів, взаємодіючих один з одним з метою надання необхідного сервісу. Таку взаємодію можна зобразити у вигляді моделі, що складається з множини абстрактних примітивів і параметрів.

Архітектура відображає основну вимогу, що лежить в основі розробки *SNMPv3,* – модульність архітектури, яка повинна:

- допускати реалізацію в широкому діапазоні операційних середовищ,

коли в одних випадках можуть передбачатися мінімальні за функціональними можливостями і вартістю рішення, а в інших – підтримка додаткових функціональних можливостей керування великими мережами;
- дозволяти просувати по шляху стандартизації окремі блоки архітектури, якщо не буде досягнуто консенсусу для всієї структури в цілому;
- допускати можливість використання альтернативних моделей захисту.

5.4.2 Об'єкт *SNMP*

Кожен об'єкт *SNMP* включає свій процесор *SNMP (SNMP engine).* Такий процесор реалізовує можливості відправлення і приймання повідомлень, їхньої автентифікації, шифрування/дешифрування, а також управління доступом до керованих об'єктів. Відповідні функції надаються у вигляді сервісів одному або декільком додаткам, побудованим для роботи з процесором *SNMP*, і формують так званий об'єкт *SNMP.*

І процесор *SNMP*, і підтримувані ним додатки визначаються у вигляді сукупності окремих модулів. Така архітектура має цілий ряд переваг. По-перше, роль об'єкта *SNMP* визначається тими модулями, що реалізовані в рамках цього об'єкта. При цьому для агента *SNMP* потрібно один набір модулів, а для адміністратора *SNMP* – інший (хоча ці набори і перекриваються). По-друге, модульна структура специфікацій виливається у відповідну гнучкість при визначенні різних версій кожного модуля. Це, у свою чергу, дозволяє визначити альтернативні або поліпшені можливості для частин *SNMP* без необхідності переходу до нової версії всього стандарту в цілому (наприклад, до *SNMPv4*) і зробити зрозумілими стратегії переходу до нових версій і співіснування таких версій.

Щоб зрозуміти роль кожного модуля і їхнього взаємозв'язку, найкраще розглянути використання цих модулів у рамках стандартних адміністраторів і агентів *SNMP.*

На рис. 5.6 показана блок-схема стандартного адміністратора *SNMP.*

Стандартний адміністратор *SNMP* взаємодіє з агентами *SNMP* за допомогою відправлення команд *(get, set)* і прийому повідомлень переривань *(trap).* Адміністратор може також відправляти іншим адміністраторам *PDU* із запитами *InformRequest,* які несуть повідомлення, і одержувати *PDU* з відповідями *InformResponse,* які підтверджують прийом повідомлень *InformRequest.* Відповідно до опису *SNMPv3* стандартний адміністратор *SNMP* включає три категорії додатків. Додаток генератора команд здійснює моніторинг і маніпулює даними керування віддалених агентів (при цьому використовуються *PDU SNMPv1* і/або *SNMPv2,* включаючи команди *Get, GetNext, GetBulk і Set).* Додаток ініціатора повідомлень генерує асинхронні повідомлення (стандартний адміністратор для цього використовує *InformRequest*). Додаток одержувача повідомлень обробляє асинхронні вхідні повідомлення, до яких відносяться повідомлення *InformRequest, SNMPv2-Trap* і повідомлення *Trap SNMPv1.*

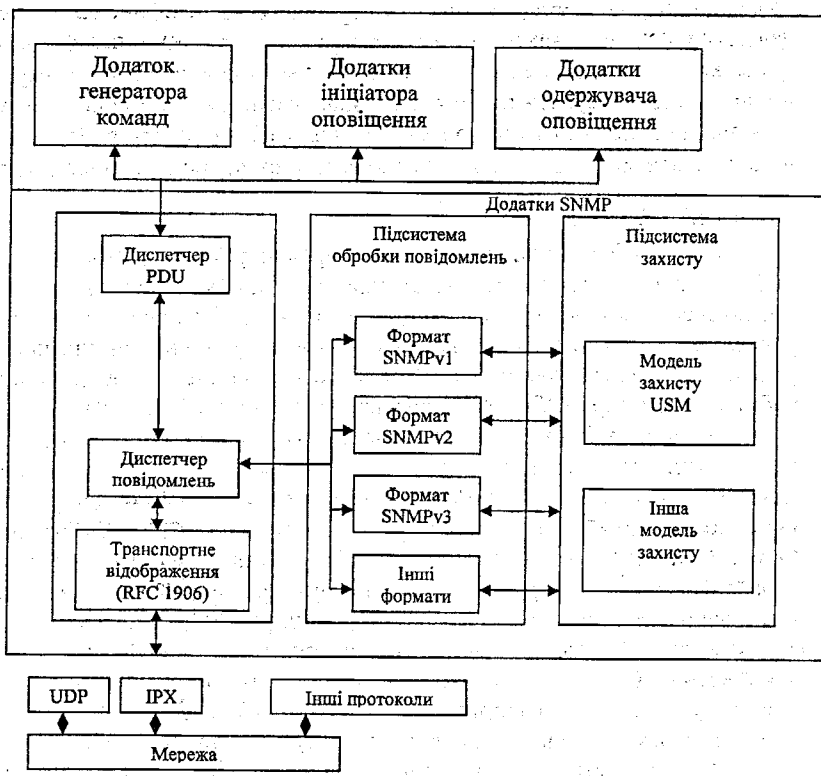


Рисунок 5.6 – Блок-схема стандартного адміністратора *SNMP*

При надходженні *PDU* типу *InformRequest* додаток одержувача повідомлень реагує відправленням *PDU* типу *Response.*

Усі згадані вище додатки використовують сервіси, надані даному об'єктові процесором *SNMP*. Процесор *SNMP* виконує такі загальні функції.

- Приймає *PDU*, що виходять від додатків *SNMP*, виконує необхідну обробку, включаючи шифрування і додавання кодів автентифікації, а потім інкапсулює *PDU* у повідомлення, які передаються далі.
- Приймає повідомлення *SNMP*, які надходять з транспортного рівня, виконує необхідну обробку, включаючи автентифікацію і дешифрування, потім витягає *PDU* з повідомлень і передає його відповідному додаткові *SNMP*.

У стандартному адміністраторові процесор *SNMP* містить диспетчер, підсистему обробки повідомлень і підсистему захисту. Диспетчер являє собою просту програму керування трафіком, що приймає *PDU*, які виходять від додатків, і виконує такі функції. Для *PDU* диспетчер визначає тип

необхідної обробки *(SNMPv1, SNMPv2* або *SNMPv3)* і відправляє *PDU* далі відповідному модулеві обробки повідомлень у підсистемі обробки повідомлень. Ця підсистема повертає повідомлення, що містить *PDU* і включає відповідні заголовки. Потім диспетчер відображає це повідомлення в транспортний рівень для передачі.

5.4.3 Стандартний адміністратор *SNMP*

Повідомлення, що надходять, обробляються за аналогічним сценарієм. Диспетчер приймає повідомлення з транспортного рівня і виконує такі функції: посилає повідомлення відповідному модулеві обробки повідомлень, підсистема обробки повідомлень повертає *PDU* з повідомлення, диспетчер передає *PDU* відповідному додаткові.

Підсистема обробки повідомлень приймає вихідні *PDU* від диспетчера і готує їх до передачі, відправляючи відповідним заголовком повідомлення і повертаючи його диспетчерові. Підсистема обробки повідомлень приймає від диспетчера вхідні повідомлення, обробляє всі заголовки і повертає диспетчерові вкладені *PDU.* Реалізація підсистеми обробки повідомлень може підтримувати формат повідомлень, що відповідає одній з версій *SNMP (SNMPv1, SNMPv2c, SNMPv3),* або містить кілька модулів, кожний з яких підтримує свою версію *SNMP.*

Підсистема захисту виконує функції автентифікації і шифрування. Вихідне повідомлення передається підсистемі захисту підсистемою обробки повідомлень. В залежності від необхідного сервісу підсистема захисту може шифрувати вкладені *PDU* та деякі поля/заголовки повідомлення, а також генерувати код автентифікації, додаючи останній у заголовок повідомлення. Оброблене повідомлення потім повертається підсистемі обробки повідомлень. Точно так само підсистема обробки повідомлень передає підсистемі захисту кожне вхідне повідомлення. Якщо необхідно, підсистема захисту перевіряє код автентифікації і виконує дешифрування, а потім повертає оброблене повідомлення підсистемі обробки повідомлень. Реалізація підсистеми захисту може підтримувати одну або кілька моделей захисту. Дотепер єдиною визначеною моделлю захисту залишається модель *USM.*

На рис. 5.7 показана блок-схема стандартного агента *SNMP.*

Стандартний агент може містити три типи додатків. Додатки модуля відповіді на команди забезпечують доступ до даних керування. Ці додатки реагують на запити, що надходять, витягаючи і/або встановлюючи значення керованих об'єктів з таким генеруванням *PDU* типу *Response.* Додаток ініціатора повідомлень генерує асинхронні повідомлення (для стандартного агента це будуть *PDU* типу *SNMPv2-Trap* або *Trap SNMPv1).* Додаток посередника передачі передає дані від об'єкта до об'єкта.

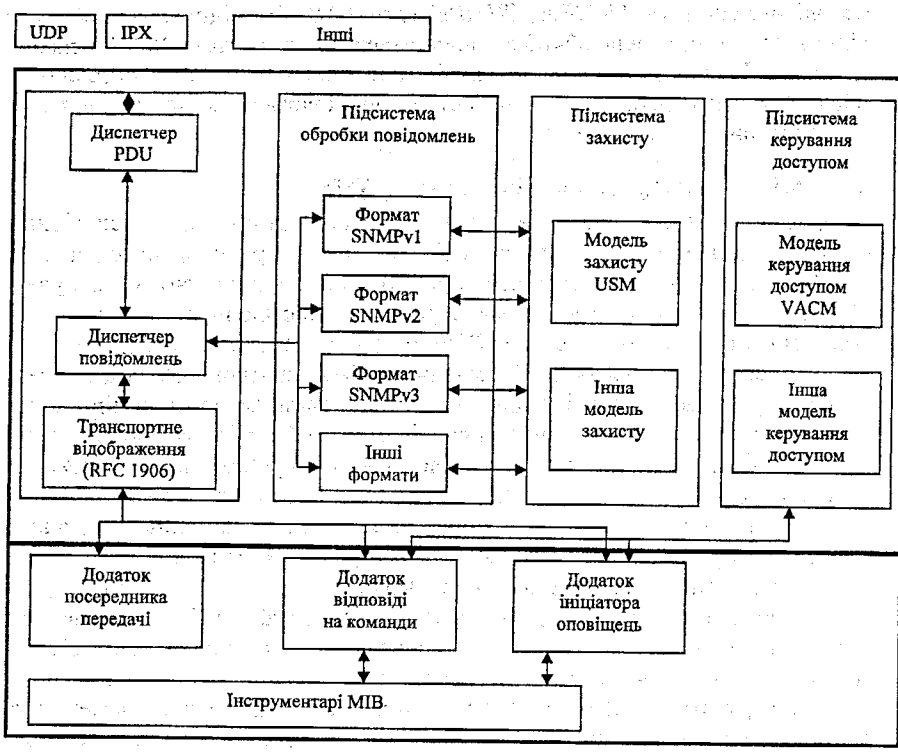


Рисунок 5.7 – Блок-схема стандартного агента SNMP

Останній додаток забезпечує сервіс авторизації для контролю доступу до структур *MIB* при читанні й установленні параметрів об'єктів керування. Усі служби використовуються на основі вмісту *PDU*. Реалізація підсистеми керування доступом може підтримувати одну або кілька моделей керування доступом. Дотепер єдиною визначеною моделлю керування доступом є модель *VACM (View-Based Access Control Model* – модель керування доступом на базі подань).

Пов'язані із захистом функції організовані в дві окремі підсистеми – захисту і керування доступом. Ці дві підсистеми виконують зовсім різні функції, тому має сенс здійснювати стандартизацію цих областей незалежно. Підсистема захисту стосується конфіденційності й автентифікації й оперує з повідомленнями *SNMP*. Підсистема керування доступом стосується авторизації доступу до інформації системи керування й оперує з протокольними одиницями обміну *SNMP*.

5.4.4 Термінологія

У табл. 5.2 подані деякі елементи *SNMP*.

Таблиця 5.2 – Ідентифікатори *SNMPv3*

Ідентифікатор	Опис
snmpEngineID	Унікальний ідентифікатор процесора *SNMP*, а також відповідного йому об'єкта *SNMP*. Визначається текстовим правилом із синтаксисом *Octet String*.
contextEngineID	Однозначно ідентифікує об'єкт *SNMP*, здатний розпізнати екземпляр контексту з конкретним ім'ям *contextName*.
contextNam	Ідентифікує конкретний контекст у рамках процесора *SNMP*. Передається як параметр диспетчерові і підсистемі керування доступом.
scopedPDU	Блок даних, що складається з *contextEngineID*, *contextName* і протокольної одиниці обміну *SNMP*. Передається як параметр до або від підсистеми захисту.
snmpMessageProcessing Model	Унікальний ідентифікатор моделі обробки повідомлень підсистеми обробки повідомлень. Можливими значеннями є *SNMPvl*, *SNMPv2c* і *SNMPv3*. Визначається текстовим правилом із синтаксисом *Integer*.
snmpSecurity Model	Унікальний ідентифікатор моделі захисту підсистеми захисту. Можливими значеннями є *SNMPvl*, *SNMPv2c* і *USM*. Визначається текстовим правилом із синтаксисом *Integer*.
snmpSecurity Level	Ступінь захисту відісланих повідомлень *SNMP* або виконуваних операцій, виражена в термінах наявності або відсутності вимоги автентифікації і/або забезпечення конфіденційності. Варіантами значень є *noAuthnoPriv*, *authNoPriv* і authPriv. Визначається текстовим правилом із синтаксисом *Integer*.
principal	Об'єкт, в інтересах якого надається сервіс або виконується обробка. Принципал може бути індивідуальним користувачем, що відіграє конкретну роль, множиною індивідуумів, кожний з яких відіграє свою роль, додатком або набором додатків, а також будь-якою комбінацією зазначених можливостей.
securityName	Поданий принципалом рядок, який сприймається людиною. Передається як параметр усім примітивам *SNMP* (диспетчерові, обробловачеві повідомлень, захистові, керуванню доступом).

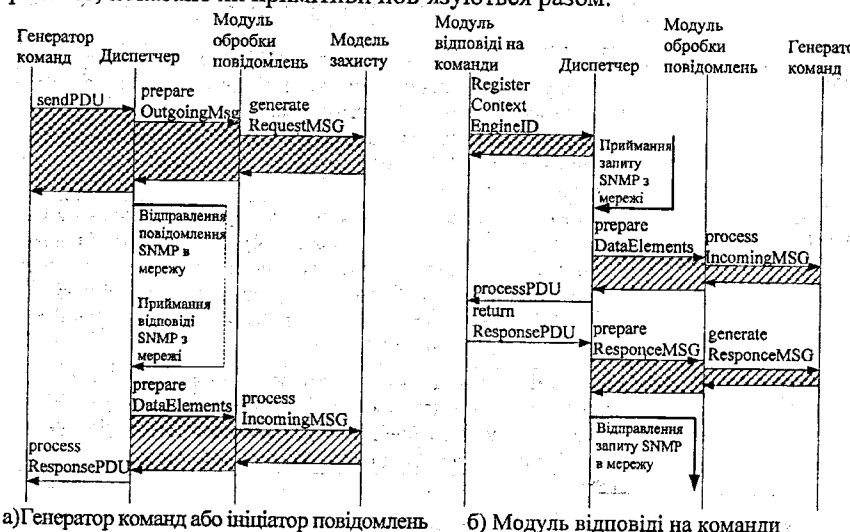
З кожним об'єктом *SNMP* пов'язується унікальний ідентифікатор *snmpEngineID*. Для керування доступом передбачається, що кожен об'єкт *SNMP* керує деяким набором контекстів інформації керування, кожний з яких має своє ім'я *contextName*, унікальне в рамках відповідного об'єкта. Щоб підкреслити, що в рамках кожного об'єкта існує окремий

адміністратор контекстів, кожен об'єкт пов'язується з унікальним ідентифікатором *contextEngineID*, а оскільки процесор контекстів однозначно відповідає процесорові *SNMP* даного об'єкта, значення *contextEngineID і snmpEngineID* ідентичні. Керування доступом здійснюється в залежності від конкретного контексту, до якого намагаються одержати доступ, і від користувача, що вимагає такий доступ. Користувач може бути як особою, так і додатком або групою осіб/додатків.

Інші важливі терміни пов'язані з обробкою повідомлень. Так, *snmpMessageProcessingModel* визначає формат повідомлення і версію *SNMP* для обробки повідомлення, *snmpSecurityModel* визначає модель захисту, а *snmpSecurityLevel* – сервіс захисту, для даної конкретної операції. Користувач може запросити або тільки автентифікацію, або автентифікацію разом з шифруванням, або нічого.

5.4.5 Додаток *SNMPv3*

Сервіс для модулів усередині об'єкта *SNMP* визначається в документах у термінах примітивів і параметрів. Примітив визначає функцію для виконання, а параметри використовуються для передачі даних і контролю інформації. Ці примітиви і параметри можна розглядати як формальний спосіб визначення сервісу *SNMP*. Конкретна форма примітива залежить від реалізації – наприклад, це може бути виклик процедури. На рис. 5.8, показано як примітиви пов'язуються разом.



а)Генератор команд або ініціатор повідомлень б) Модуль відповіді на команди

Рисунок 5.8 – Пов'язування примітивів

На рис. 5.8, *а* показана послідовність подій, у результаті настання яких додатки генератора команд або ініціатора повідомлень запитують

відправлення *PDU*. Згодом відповідним додаткам повертаються відповіді (ці події відбуваються в адміністраторі). На рис. 5.8, *б* показані події, що відбуваються в системі агента. Ілюстрація пояснює, як *PDU* з вхідного повідомлення переправляється диспетчером додатку, і як відповідь додатка відображається у вихідному повідомленні.

На рис. 5.8 деякі стрілки діаграми мають ярлики з іменами примітивів, що зображають виклик. Непомічені стрілки зображають відповіді на ці виклики, а затінення вказує на відповідність між викликом і відповіддю.

5.4.6 Потік *SNMPv3*

Додаток генератора команд використовує примітиви диспетчера *SendPdu* та *processResponsePdu*. При цьому *SendPdu* забезпечує диспетчера інформацією про адресата, параметри захисту і *PDU*. Щоб підготувати повідомлення, диспетчер викликає модель обробки повідомлень, що, у свою чергу, викликає модель захисту. Диспетчер передає підготовлене повідомлення на транспортний рівень (наприклад *UDP*) для передачі. Якщо підготовка повідомлення завершується аварійно, повернуте значення примітива *sendPdu*, установлене диспетчером, вказує на помилку. Якщо підготовка повідомлення завершується успішно, диспетчер приписує *PDU* ідентифікатор *sendPduHandle* і повертає це значення генераторові команд. Генератор команд зберігає *sendPduHandle*, щоб згодом його можна було порівняти з *PDU* відповіді на вихідний запит.

Диспетчер доставляє *PDU* відповідей відповідним додаткам генератора команд, використовуючи примітив *processResponsePdu*.

Додаток відповіді на команди використовує чотири примітиви диспетчера *(registerContextEngineID, unregisterContextEngineID, processPdu, returnResponsePdu)* і один примітив підсистеми керування доступом *(isAccessAllowed)*.

Примітив *registerContextEngineID* дозволяє додаткові відповіді на команди асоціювати з процесором *SNMP* з метою обробки *PDU* визначених типів для процесора контекстів. Після реєстрації модуля відповіді на команди, всі асинхронно отримані повідомлення, що містять відповідну комбінацію підтримуваних *contextEngineID* і *pduType*, відсилаються модулеві відповіді на команди. Модуль відповіді на команди може скасувати асоціацію з процесором *SNMP*, використовуючи примітив *unregisterContextEngineID*.

Диспетчер доставляє вхідні *PDU* запитів відповідному додаткові відповіді на команди, використовуючи примітив *processPdu*. Потім модуль відповіді на команди виконує такі кроки.

- Модуль відповіді на команди перевіряє *PDU* запиту. Тип операції повинен відповідати одному з типів, зареєстрованих цим додатком.
- Модуль відповіді на команди визначає, чи дозволений доступ до опе-

рації керування, яка запитується *PDU*. Для цього викликається примітив *isAccessAllowed*. Параметр *securityModel* указує, яку модель захисту повинна використовувати підсистема керування доступом, відповідаючи на цей виклик. Підсистема керування доступом визначає, чи має примітив *(securityName)* на цьому рівні захисту *(securityLevel)* право запросити дану операцію керування *(viewType)* щодо керованого об'єкта *(variableName)* у даному контексті *(contextName)*.

- Якщо доступ дозволений, модуль відповіді на команди виконує операцію керування і генерує *PDU* відповіді. Якщо при доступі виникає помилка, генерується *PDU*, що сигналізує про помилку.
- Модуль відповіді на команди за допомогою примітива *returnResponsePdu* викликає диспетчера для відправлення *PDU* відповіді.

Додаток генератора повідомлень відповідає тому ж наборові загальних процедур, що і додаток генератора команд. Якщо необхідно послати *PDU* типу *Inform Request* (інформація запиту), використовуються примітиви *sendPdu* і *processResponsePdu*, так само, як це було в додатку генератора команд. Якщо необхідно послати *PDU* типу переривання, використовується тільки примітив *sendPdu*.

Додаток одержувача повідомлень додержується підмножини набору загальних процедур, використовуваному додатком модуля відповіді на команди. Одержувач повідомлень повинен спочатку зареєструватися для одержання *PDU* типу *Inform* (інформація) і/або *trap* (переривання). Обидва типи *PDU* приймаються за допомогою примітива *processPdu*. У *PDU* типу *Inform* для відповіді використовується примітив *returnResponsePdu*.

Додаток посередника передачі використовує примітиви диспетчера для передачі повідомлень *SNMP* далі. Посередник передавання обробляє такі чотири типи повідомлень.

- Повідомлення, що містять типи *PDU* додатка генератора команд. Посередник передавання визначає, чи є процесор *SNMP* кінцевим одержувачем або це процесор *SNMP*, найближчий до кінцевого одержувача, і посилає *PDU* запиту відповідного виду.
- Повідомлення, що містять типи *PDU* додатка ініціатора повідомлень. Посередник передачі визначає, які процесори *SNMP* повинні одержати дане повідомлення і посилає *PDU* повідомлення певного виду.
- Повідомлення, що містить *PDU* типу *Response* (відповідь). Посередник передавання визначає, яке з раніше переданих повідомлень запиту відповідає даній відповіді, і посилає *PDU* відповіді відповідного виду.
- Повідомлення, що містять указівки звіту. *PDU* типу *Report* (звіт) використовуються в *SNMPv3* для обміну даними між процесорами. Посередник передачі визначає, яке з раніше переданих повідомлень запиту відповідає даній указівці звіту, і передає вказівку звіту джерелу запиту або повідомлення.

5.5 Модель обробки повідомлень і захисту користувача

Для обробки повідомлень використовуються модель обробки повідомлень загального призначення і конкретна модель захисту. Їхній зв'язок показаний на рис. 5.8.

5.5.1 Модель обробки повідомлень

Ця модель відповідає за прийом *PDU* від диспетчера, інкапсуляцію *PDU* у повідомлення і виклик *USM* (модель захисту користувача) для додавання в заголовок повідомлення параметрів, пов'язаних із захистом. Модель обробки повідомлень також приймає вхідні повідомлення, викликає *USM* для обробки пов'язаних із захистом параметрів у заголовку повідомлення і доставляє інкапсульовані *PDU* диспетчерові.

На рис. 5.9 показана структура повідомлення. Перші п'ять полів генеруються моделлю обробки повідомлень для вихідних повідомлень і обробляються моделлю обробки повідомлень для вхідних. Таких шість полів указують параметри захисту, які використовуються у *USM*. Нарешті, *PDU* разом зі значеннями *contextEngineID* і *contextName* складають область, яка використовується при обробці *PDU*.

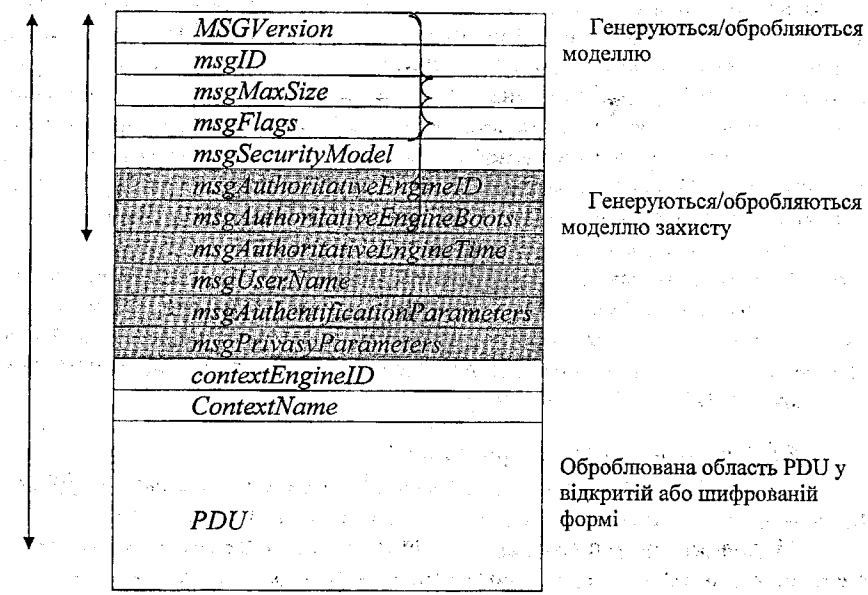


Рисунок 5.9 – Формат повідомлення SNMPv3, що використовує USM

Перші п'ять полів такі.

msgVersion. Установлюється рівним *SNMPv3 (3).*

msgID. Унікальний ідентифікатор, який використовується двома об'єктами *SNMP* для координації повідомлень запиту і відповіді, а також процесором повідомлень для координації обробки повідомлення різними моделями підсистем у рамках наявної архітектури. Значення ідентифікатора знаходяться в діапазоні від 0 до $(2^{31}-1)$.

msgMaxSize. Указує максимальний розмір повідомлення в байтах, який підтримується відправником повідомлення, з областю значень від 484 до $(2^{31}-1)$. Це максимальний розмір сегмента, що відправник може прийняти від іншого процесора *SNMP* (у відповіді або в повідомленні іншого типу).

msgFlags. Рядок байтів, що містить три прапорці в трьох молодших розрядах: *reportableFlag, privFlag, authFlag.* Якщо *reportableFlag* = 1, то відправникові повертається *PDU* типу *Report* (звіт). Якщо цей прапорець дорівнює нулеві, то *PDU* типу *Report* може не посилатися. Відправник установлює *ReportableFlag* рівним 1 у повідомленнях, що містять запит *(Get, Set),* і повідомленнях типу *Inform*, і рівним 0 у повідомленнях, що містять *PDU* типу *Response* (відповідь), *Trap* (переривання) або *Report.* Прапорець *ReportableFlag* є допоміжним засобом визначення необхідності відправлення повідомлення типу *Report.* Він використовується тільки тоді, коли порцію *PDU* повідомлення декодувати не вдається (наприклад, коли дешифрування завершується невдачею через неправильний ключ). Прапорці *PrivFlag* і *authFlag* використовуються відправником для вказання ступеня захисту повідомлення. Значення *privFlag* = 1 говорить про застосування засобів шифрування, а значення *authFlag* = 1 – про використання засобів автентифікації. Дозволяються всі комбінації, крім *privFlag = 1 AND authFlag = 0,* тобто шифрування без автентифікації не допускається.

msgSecurityModel. Ідентифікатор з областю значень від 0 до $(2^{31}-1)$, що вказує модель захисту, яка застосовувалася відправником при підготовці цього повідомлення, і, отже, модель захисту, яку варто використовувати одержувачеві для обробки цього повідомлення. Зарезервованими значеннями є 1 для *SNMPv1*, 2 для *SNMPv2c* (*SNMPv2* з можливостями об'єднань *SNMPv1*) і 3 для *SNMPv3.*

5.5.2 Модель захисту користувача

USM забезпечує сервіс автентифікації та таємності в рамках *SNMP.* Модель *USM* розроблялася з метою захисту від загроз таких типів.

Модифікація інформації. По шляху проходження повідомлення, згенерованого авторизованим об'єктом, зловмисник може змінити це повідомлення, щоб виконати несанкціоновані операції керування (наприклад, установивши відповідні значення об'єкта керування). Суть загрози полягає в тому, що зловмисник може змінити будь-які параметри керування, включаючи параметри конфігурації, виконуваних дій і контролю.

Імітація. Зловмисник може намагатися виконати недозволені йому

операції керування, ототожнюючи даний об'єкт із деяким авторизованим.

Модифікація потоку повідомлень. Протокол *SNMP* призначений для роботи над транспортним протоколом, який не передбачає встановлення з'єднання. Існує загроза переупорядкування, затримки або відтворення (дублювання) повідомлень *SNMP* для несанкціонованого керування. Наприклад, можна скопіювати і, згодом, відтворити повідомлення, яке викликає перезавантаження пристрою.

Розголошення інформації. Спостерігаючи за потоком обміну даними між адміністратором і агентом, зловмисник може з'ясувати значення керованих об'єктів і розпізнати події, які підлягають реєстрації. Наприклад, спостереження за набором команд, які змінюють паролі, може дозволити довідатися про нові паролі.

Модель USM не передбачає захист від загроз таких двох типів.

Блокування сервісу. Зловмисник може не допустити обмін даними між адміністратором і агентом.

Аналіз трафіка. Зловмисник може аналізувати загальну структуру трафіка між адміністраторами та агентами.

Відсутність засобів протистояння загрозі блокування сервісу витікає з таких двох причин. По-перше, атаки блокування сервісу в багатьох випадках не відрізняються від відмовлень мережі, з якими звичайно приходиться мати справу будь-якому реальному додаткові мережевого керування. По-друге, атака блокування сервісу, імовірніше всього, розірве всі типи обміну даними, тобто така атака повинна бути проблемою загальних засобів захисту, а не тільки протоколу мережевого керування.

Що стосується аналізу трафіка, то цілий ряд структур такого трафіка цілком прогнозований (наприклад, об'єкти можуть керуватися командами *SNMP,* що генеруються з певною регулярністю однією або декількома станціями керування), тому немає особливої необхідності захищатися від можливості спостереження за цими структурами.

5.5.3 Криптографічні функції

В моделі *USM* використовуються дві криптографічні функції – автентифікація та шифрування. Для підтримки цих функцій процесорові *SNMP* потрібні ключ таємності *(privKey)* і ключ автентифікації *(authKey).* Підтримка значень цих двох ключів здійснюється як для локальних, так і віддалених користувачів.

Ці значення є атрибутами користувача, що зберігаються. Значення *privKey* та *authKey* недоступні в рамках засобів *SNMP.*

Модель *USM* допускає використання одного з двох альтернативних протоколів автентифікації – *HMAC-MD5-96* або *HMAC-SHA-96.* Алгоритм *HMAC,* використовує захищену функцію хешування і секретний ключ для одержання коду автентичності повідомлення. У *HMAC-MD5-96* як функція хешування використовується *MD5,* а на вхід алгоритму подається 16-

байтове (128-бітове) значення *authKey*. Алгоритм породжує 128-бітовий результат, який усікається до 12 байт (96 біт). У *HMAC-SHA*-96 функцією хешування є *SHA-1*, а значення *authKey* має довжину 20 байт. Алгоритм породжує 20-байтовий результат, який теж усікається до 12 байт.

У моделі *USM* для шифрування використовується режим *CBC* (режим зчеплення шифрованих блоків) алгоритму *DES*. На вхід протоколу шифрування подається 16-байтове значення *privKey*. Перші 8 байт (64 біт) *privKey* використовуються як ключ *DES*. Оскільки для *DES* потрібен 56-бітовий ключ, молодший розряд кожного байта ігнорується. Для режиму *CBC* потрібен 64-бітовий вектор ініціалізації. Останні 8 байт *privKey* містять значення, використовуване для генерування вектора ініціалізації.

5.5.4 Авторитетні і неавторитетні процесори

При будь-якому передаванні повідомлень один із двох об'єктів, відправник або одержувач, розглядається як авторитетний процесор *SNMP* відповідно до таких правил.

- Якщо повідомлення *SNMP* містить корисний вантаж, для якого потрібна відповідь (наприклад, *PDU* типу *Get, GetNext, GetBulk, Set* або *Inform*), то авторитетним є одержувач повідомлення.
- Якщо повідомлення *SNMP* містить корисний вантаж, для якого не очікується відповіді (наприклад, *PDU* типу *SNMPv2-Trap, Response* або *Report*), то авторитетним є відправник повідомлення.
- Для повідомлень, що відсилаються від імені генератора команд, і для повідомлень *Inform* ініціатора повідомлень авторитетним процесором є одержувач.
- Для повідомлень, що відсилаються від імені модуля відповіді на команди, і для повідомлень *Trap* ініціатора повідомлень авторитетним процесором є відправник.

Своєчасність повідомлення визначається в порівнянні з годинами, підтримуваними системою авторитетного процесора. Коли авторитетний процесор посилає повідомлення (*Trap*, *Response*, *Report*), воно містить поточне значення його часу, щоб неавторитетний одержувач міг синхронізувати свій час з цим годинником. Коли неавторитетний процесор посилає повідомлення (*Get*, *GetNext*, *GetBulk*, *Set*, *Inform*), він включає в повідомлення оцінку поточного значення часу в точці призначення, що дозволяє оцінити своєчасність доставки повідомлення.

Процес локалізації ключів, дає можливість одному користувачу володіти ключами, збереженими множиною процесорів; ці ключі локалізуються авторитетним процесором таким чином, щоб користувач відповідав тільки за один ключ, уникаючи необхідності зберігати декілька екземплярів одного і того ж ключа в розподіленій мережі.

Одержувач *PDU* генератора команд і *PDU* типу *Inform* вважається авторитетним процесором, а отже, і відповідальним за перевірку

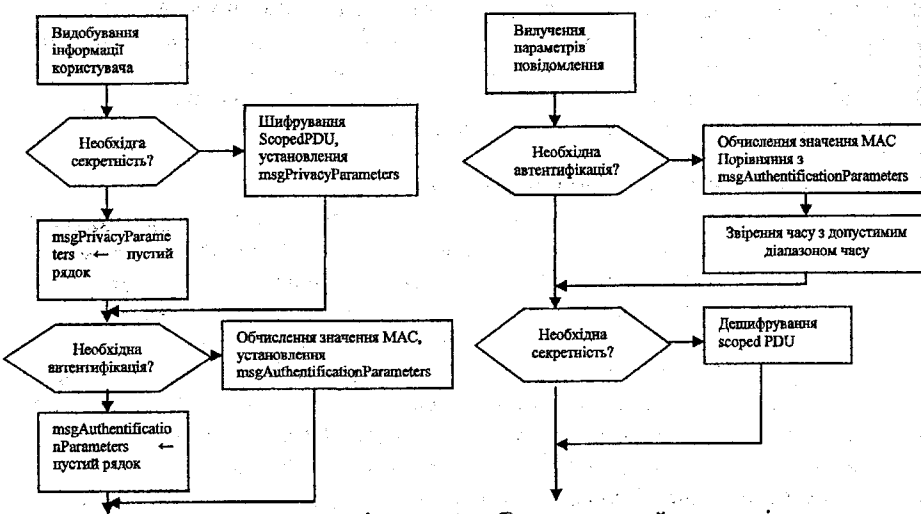
своєчасності доставки повідомлень. Якщо відповідь або переривання виявляються затриманими або відтвореними, небезпека не занадто велика. Але *PDU* генератора команд і, частково, *PDU* типу *Inform* ініціюють операції керування – наприклад, читання або установлення значень об'єктів *MIB*. Тому важливо гарантувати, що такі *PDU* не затримуються і не відтворюються, щоб виключити несанкціоновані дії.

5.5.5 Параметри повідомлень *USM*

Коли процесор повідомлень передає підсистемі *USM* вихідне повідомлення, підсистема *USM* заповнює пов'язані із захистом параметри в заголовку повідомлення. Коли процесор повідомлень передає підсистемі *USM* вхідне повідомлення, *USM* обробляє значення, яке зберігається у відповідних полях. Із захистом пов'язані такі параметри.

- *msgAuthoritativeEnginel.* Значення *snmpEngineID* авторитетного процесора *SNMP*, задіяного в процесі обміну даним повідомленням. Це значення посилається на джерело в повідомленнях *Trap, Response* або *Report*, на адресата – у *Get, GetNext, GetBulk, Set* або *Inform.*
- *msgAuthoritativeEngineBoots.* Значення *snmpEngineBoots* авторитетного процесора *SNMP*, задіяного в процесі обміну даним повідомленням. Об'єкт *snmpEngineBoots* є цілим числом з діапазону від 0 до $(2^{31}-1)$, і зображає число ініціалізацій та реініціалізацій даного процесора *SNMP* з моменту установлення початкової конфігурації.
- *msgAuthoritativeEngineTinie.* Значення *snmpEngineTime* авторитетного процесора *SNMP*, задіяного в процесі обміну даним повідомленням. Об'єкт *snmpEngineTime* є цілим числом з діапазону від 0 до $(2^{31}-1)$, і зображає час у секундах, який пройшов з моменту останнього збільшення значення об'єкта *snmpEngineBoots* даним авторитетним процесором *SNMP*. Кожен авторитетний процесор *SNMP* відповідає за збільшення свого значення *snmpEngineTime* раз у секунду. Неавторитетний процесор відповідає за збільшення свого подання *snmpEngineTime* для кожного віддаленого авторитетного процесора, з яким відкритий зв'язок.
- *msgUserName.* Користувач, від імені якого відбувається обмін повідомленнями.
- *msgAuthenticationParameters.* Не містить значень, якщо автентифікація для цього процесу обміну не використовується. В протилежному випадку містить параметр автентифікації, тобто код автентичності повідомлення *HMAC* (відповідно до *USM*).
- *msgPrivacyParameters.* Не містить значень, якщо для цього процесу обміну не використовуються засіб забезпечення таємності (шифрування). В протилежному випадку містить параметр таємності, тобто значення, використовуване для генерування початкового значення вектора ініціалізації у режимі *CBC* алгоритму *DES.*

На рис. 5.10 показана схема функціонування *USM*.



а) алгоритм передавання повідомлення б) алгоритм приймання повідомлення

Рисунок 5.10 – Схема функціонування USM

Перед відправленням повідомлення, якщо це необхідно, виконується шифрування. Оброблювана область *PDU* шифрується і розміщується в блоці корисного вантажу повідомлення, а значення *msgPrivacyParameters* установлюється рівним значенню, яке потрібне для генерування вектора ініціалізації. Потім при необхідності виконується автентифікація. Усе повідомлення, включаючи оброблювану область *PDU*, подається на вхід алгоритма *HMAC*, а отриманий у результаті код автентичності міститься в *msgAuthenticationParameters*.

Для вхідних повідомлень спочатку, якщо необхідно, виконується автентифікація. Система *USM* порівнює значення *MAC* з обчисленим значенням *MAC* і, якщо ці два значення збігаються, повідомлення вважається автентифікованим.

Потім *USM* перевіряє, чи попадає повідомлення в рамки припустимого вікна часу. Якщо повідомлення виявляється невідповідним часу, воно відкидається як фальсифіковане. Нарешті, якщо оброблювана область *PDU* була зашифрована, *USM* дешифрує і повертає відкритий текст.

5.6 Обробка повідомлення в моделі *USM*

5.6.1 Механізми контролю часу *USM*

Модель *USM* включає набір механізмів контролю часу для протистояння спробам затримки і відтворення повідомлень. Кожен процесор *SNMP*, що може діяти як авторитетний процесор, повинен підтримувати два

об'єкти, *snmpEngineBoots* і *snmpEngineTime*, які посилаються на локальний час процесора. Коли процесор *SNMP* ініціалізується, значення цих двох об'єктів установлюються рівними нулю. Надалі значення *snmpEngineTime* постійно збільшується на одиницю в секунду. Коли *snmpEngineTime* досягає максимального значення (2^{31}-1), збільшується значення *snmpEngineBoots*, як при перезапуску системи, а значення *snmpEngineTime* установлюється рівним нулю і починає збільшуватися знову. Використовуючи механізм синхронізації, неавторитетний процесор оцінює значення часу для кожного авторитетного процесора, з яким необхідно підтримувати зв'язок. Ці оцінні значення містяться в кожному вихідному повідомленні і дозволяють приймальному авторитетному процесорові визначити, вчасно надійшло повідомлення чи ні.

Механізм синхронізації працює в такий спосіб. Неавторитетний процесор зберігає локальну копію таких трьох змінних для кожного відомого йому авторитетного процесора *SNMP*.

- *snmpEngineBoots.* Останнє значення *snmpEngineBoots* для віддаленого авторитетного процесора.
- *snmpEngineTime.* Подання *snmpEngineTime* даного процесора для віддаленого авторитетного процесора. Це значення синхронізується з віддаленим авторитетним процесором. Між подіями синхронізації це значення автоматично збільшується на одну одиницю в секунду для приблизної синхронізації з віддаленим авторитетним процесором.
- *latestReceivedEngineTime.* Найбільше *msgAuthoritativeEngineTime,* отримане даним процесором від віддаленого авторитетного процесора. Це значення оновлюється кожного разу, коли збільшується значення *msgAuthoritativeEngineTime.* Призначенням цієї змінної є захист від атак відтворення повідомлень.

Кожному процесорові необхідно підтримувати по одному набору цих трьох змінних для кожного віддаленого авторитетного процесора, з якими існує зв'язок. На практиці для цих значень виділяється свого роду хеш, де дані індексуються за унікальним значенням *snmpEngineID* віддалених авторитетних процесорів.

Щоб надати неавторитетним процесорам можливість синхронізувати час, авторитетні процесори встановлюють свої поточні значення завантаження *(boot)* і часу *(time)* разом зі значенням ідентифікатора *snmpEngineID* у поле *msgAuthoritativeEngineBoots*, поле *msgAuthoritativeEngineTime* і поле *msgAuthoritativeEngineID* кожного вихідного повідомлення типу *Response; Report* або *Trap.* Якщо повідомлення визнається справжнім і попадає в рамки припустимого вікна часу, то приймальні неавторитетні процесори обновляють свої локальні змінні *(snmpEngineBoots, snmpEngineTime і latestReceivedEngineTime)* для відповідного віддаленого процесора за такими правилами.

Відновлення відбувається, якщо виконується принаймні одна з двох умов:

- *(msgAuthoritativeEngineBoots > snmpEngineBoots)*
- *[(msgAuthoritativeEngineBoots=snmpEngineBoots)*ТА *(msgAuthoritativeEngineTime > latestReceivedEngineTime)]*

Перша умова говорить про те, що відновлення повинне бути виконане, якщо значення числа завантажень, отримане від авторитетного процесора, збільшилося з часу останнього відновлення. Друга умова говорить про те, що якщо значення числа завантажень не зросло, то відновлення повинно бути виконане, якщо значення часу, що надійшло, більше значення часу, отриманого безпосередньо перед цим. Значення часу, що надійшло, буде менше попереднього, якщо повідомлення прибудуть у неправильному порядку (що цілком можливо) або при атаці відтворення. У будь-якому випадку приймальний процесор відновлення виконувати не буде.

У результаті виклику процедури відновлення відбуваються такі зміни:

- значення *snmpEngineBoots* установлюється рівним значенню *msgAuthoritativeEngineBoots,*
- значення *snmpEngineTime* установлюється рівним значенню *msgAuthoritativeEngineTime,*
- значення *latestReceivedEngineTime* установлюється рівним значенню *msgAuthoritativeEngineTime.*

Відновлення не відбувається, при виконанні умови *msgAuthoritativeEngineBoots < snmpEngineBoots.* Таке повідомлення вважається сфальсифікованим і повинно ігноруватися.

Якщо виконується *msgAuthoritativeEngineBoots = snmpEngineBoots,* але *msgAuthoritativeEngineTime < latestReceivedEngineTime,* то відновлення теж не відбувається. У цьому випадку повідомлення може бути справжнім, але прийшло з порушенням порядку проходження, а при цьому відновлення *snmpEngineTime* виявляється сумнівним.

Синхронізація повинна виконуватися тільки тоді, коли для повідомлення використовується сервіс автентифікації, і дійсність повідомлення перевірена за допомогою алгоритму *HMAC.* Це обмеження важливе, оскільки область дії автентифікації включає значення *msgAuthoritativeEngineID, msgAuthoritativeEngineBoots* та *msgAuthoritativeEngineTime,* гарантуючи достовірність цих значень.

SNMPv3 потребує, щоб повідомлення було доставлено з певною затримкою в часі, щоб виключалася можливість проведення атак затримки і відтворення повідомлень. Вікно часу повинно при цьому бути мінімальним з урахуванням точності наявних годинників, затримок зв'язку і частоти, з якою відбувається синхронізація часу. Якщо вікно часу занадто мале, автентичні повідомлення часто будуть відкидатися, як фальсифіковані. З іншого боку, при занадто великому вікні часу підвищується вразливість системи щодо зловмисних затримок повідомлень.

Для кожного вхідного повідомлення, що успішно пройшло

автентифікацію, у якого значення *msgAuthoritativeEnginel* збігається зі значенням *snmpEngineID* даного процесора, процесор порівнює значення *msgAuthoritativeEngineBoots* і *msgAuthoritativeEngineTime* з повідомлення, що надійшло, зі значеннями *snmpEngineBoots* і *snmpEngineTime,* підтримуваними цим процесором. Повідомлення, що надійшли, вважаються такими, що вийшли за рамки припустимого вікна часу при виконанні кожної з таких умов:

- *snmpEngineBoots* = 2^{31}-1
- *msgAuthoritativeEngineBoots* ≠ *snmpEngineBoots*
- значення *msgAuthoritativeEngineTime* відрізняється від значення *snmpEngineTime* більш ніж на ±150 секунд.

Перша умова говорить, що, якщо *snmpEngineBoots* досягає свого максимального значення, жодне з повідомлень, що надходить, не може вважатися справжнім. Друга умова говорить про те, що повідомлення повинне містити значення числа завантажень рівне значенню числа завантажень локального процесора. Наприклад, якщо локальний процесор був перезавантажений, а віддалений процесор з тих пір не був з ним синхронізований, то повідомлення такого віддаленого процесора не буде вважатися автентичним. Остання умова означає, що час у вхідному повідомленні повинен відрізнятися від локального часу не більш, ніж на 150 секунд у будь-яку сторону.

Якщо повідомлення виходить за рамки допустимого вікна часу, то вважається, що повідомлення не є справжнім, і модулеві повертається повідомлення про помилку *(notInTimeWindow).*

Як і при синхронізації, контроль часу здійснюється тільки при використанні сервісу автентифікації і за умови автентичності повідомлення, тобто гарантується достовірність полів заголовка повідомлення.

5.6.2 Локалізація ключів

Вимога використання сервісу автентифікації і таємності в *SNMPv3* таке, що для будь-якого зв'язку між користувачем у неавторитетному процесорі і віддаленим авторитетним процесором секретні ключі автентифікації і шифрування повинні використовуватися спільно. Ці ключі дають можливість користувачеві в системі неавторитетного процесора застосовувати автентифікацію і таємність у віддалених авторитетних системах, якими він керує.

Щоб спростити процес керування ключами, покладений на користувача, кожному користувачеві потрібно підтримувати тільки один ключ автентифікації й один ключ шифрування. Ці ключі не зберігаються в структурі *MIB* і не доступні в рамках *SNMP.*

Користувачеві потрібно 16-байтовий ключ таємності і 16- або 20-байтовий ключ автентифікації. Для людини бажано, щоб це були не довільні рядки бітів, а слова, які можна прочитати. Модель *USM* сама по

собі не припускає будь-яких обмежень на пароль, але локальні політики керування вимагають, щоб користувачі вибирали паролі, які нелегко угадати.

Пароль при створенні ключів використовується в такий спосіб.

- На основі пароля користувача генерується рядок байтів довжиною в 2^{20} октетів (1048576 байт) шляхом багаторазового повторення значення пароля й усікання (при необхідності) останнього значення. Цей рядок називається digest0. Наприклад, для одержання *digest0* з 8-символьного пароля (2^3 байт) необхідно зв'язати конкатенацією 2^{17} однакових значень пароля.
- Якщо потрібний 16-байтовий ключ, використовується хеш-код *MD5* рядка *digest0,* у результаті чого виходить *digest1.* Якщо ж необхідний 20-байтовий ключ, то для одержання *digest1* до рядка *digest0* застосовується алгоритм хешування *SHA-1.* Результатом при цьому є ключ користувача.

Одна з переваг цієї технології полягає в тому, що істотно збільшується час, необхідний для проведення атаки перебору усіх варіантів за словником, коли зловмисникові приходиться перевіряти багато різних потенційних паролів, генеруючи ключ для кожного з них і з'ясовуючи, чи підходить він для автентифікації або шифрування. Наприклад, перехопивши автентифіковане повідомлення, зловмисник може спробувати генерувати значення *HMAC* за допомогою різних ключів користувача. Знайшовши збіг, зловмисник може вважати, що пароль знайдений. Описаний вище процес істотно збільшує час, необхідний для такої атаки.

Інша перевага цієї технології полягає в тому, що вона відокремлює ключі користувача від системи мережевого керування. Для системи мережевого керування немає необхідності зберігати значення ключів користувача. Замість цього, ключ користувача генерується з пароля цього користувача тоді, коли це дійсно потрібно. Нижче наводиться такий список причин, з яких використання паролів, незалежних від системи мережевого керування, виявляється кращим.

- Якщо ключ приходиться зберігати, а не генерувати з пароля, то можна використовувати для цього централізоване сховище секретних ключів. Але це знижує загальну надійність системи й утруднює пошук несправностей, якщо таке сховище виявиться недоступним.
- З іншого боку, підтримка резервних сховищ послаблює захист, оскільки з'являється більше потенційних цілей для проведення атак.
- Централізовані або резервні сховища повинні розміщатися в захищених місцях. Це може створювати труднощі в "пожежних" ситуаціях (коли сегменти мережі залишаються нероботоздатними і/або недоступними протягом непрогнозованого періоду часу).

Один пароль може використовуватися для генерування і ключа автентифікації, і ключа шифрування, але більш надійною схемою є використання двох паролів: одного – для генерування ключа

автентифікації, а іншого – для ключа шифрування.

Локалізований ключ визначається як секретний ключ, який використовується спільно користувачем і одним авторитетним процесором *SNMP*. Метою є можливість для користувача підтримувати тільки один ключ (або тільки два ключі, якщо потрібні як автентифікація, так і таємність), і, отже, пам'ятати тільки один пароль (або тільки два). Секретні значення, використовувані спільно конкретним користувачем з різними авторитетними процесорами *SNMP*, у дійсності виявляються різними. Процес, у результаті якого єдиний ключ користувача перетворюється в множину унікальних ключів, по одному для кожного віддаленого процесора *SNMP*, називається локалізацією ключів.

Можна визначити такі цілі керування ключами.

- Кожна система агента *SNMP* у розподіленій мережі має власний унікальний ключ для кожного користувача, авторизованого для керування цим агентом. Якщо керування дозволене множині користувачів, агент має по унікальному ключу автентифікації й унікальному ключу шифрування для кожного користувача. Таким чином, якщо ключ одного користувача скомпрометований, ключі інших користувачів залишаться надійними.
- Ключі для одного користувача в системах різних агентів відрізняються. Тому якщо деякий агент скомпрометований, то будуть скомпрометовані тільки ключі користувача цього агента, але не ключі цього користувача для інших агентів.
- Мережеве керування може здійснюватися з будь-якої точки мережі, незалежно від доступності попередньо побудованої системи мережевого керування. Це дає користувачеві можливість виконувати функції керування з будь-якої станції керування.

Можна також відзначити такі моменти, яких варто уникати.

- Користувачеві необхідно запам'ятовувати (або обробляти якимсь іншим способом) велике число ключів, і це число росте з доданням нових агентів, якими необхідно керувати.
- Зловмисник, який дізнається ключ для одного агента, з цього моменту може імітувати будь-якого іншого агента перед будь-яким користувачем або будь-якого користувача перед будь-яким іншим агентом.

Щоб вирішити всі зазначені проблеми, єдиний ключ користувача відображається за допомогою необоротної однобічної функції (тобто, захищеної функції хешування) у різні локалізовані ключі для різних автентифікованих процесорів (різних агентів). Процедура така.

- Формується рядок *digest2* за допомогою конкатенації *digest1* (описаної раніше), значення *snmpEngineID* авторитетного процесора і digest1.
- Якщо необхідно 16-байтовий ключ, застосовується хеш-код *MD5*

рядка *digest2*. Якщо потрібний 20-байтовий ключ, використовується хеш-код *SHA-1* рядка *digest2*. Результатом є локалізований ключ користувача.

Локалізований ключ може бути потім розміщений у системі агента деяким захищеним способом. Завдяки однонаправленості *MD5* і *SHA-1* зловмисник не зможе відкрити ключ користувача, навіть з'ясувавши значення локалізованого ключа.

Схема процесу локалізації ключів показана на рис. 5.11.

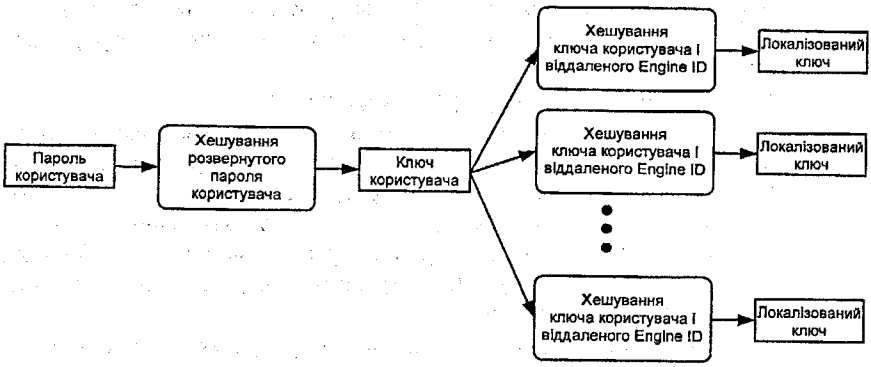


Рисунок 5.11 – Локалізація ключів

5.6.3 Керування доступом на основі подань

Керування доступом є функцією захисту, виконуваною на рівні *PDU*. Документ керування доступом задає механізми з'ясування можливості доступу до керованого об'єкта в локальній структурі *MIB* з боку віддаленого користувача. Можуть бути визначені також механізми мультидоступу. Документи *SNMPv3* визначають модель *VACM (View-Based Access Control Model* – модель керування доступом на основі подань). Модель *VACM* використовує структуру *MIB*, на основі якої визначається політика керування доступом для даного агента і стає можливим віддалене керування конфігурацією.

VACM визначає, чи дозволений віддаленому користувачу доступ до керованого об'єкта в локальній структурі *MIB*.

VACM використовує структуру *MIB* для того, щоб визначити політику керування доступом для даного агента та уможливлює використання віддаленої конфігурації.

5.6.4 Елементи моделі *VACM*

Модель *VACM*, складається зі ступеня захисту, контексту, подання *MIB* і політики доступу.

Група визначається як набір пар *<securityModel, securityName>*

(можливо порожній), у рамках якого може здійснюватися доступ до об'єктів керування *SNMP*. Значення *securityName* указує користувача, а права доступу для всіх користувачів однієї групи ідентичні. З кожною групою пов'язується унікальне значення *groupName*. Поняття групи виявляється корисним засобом класифікації адміністраторів за правами доступу. Наприклад, адміністратори вищого рівня можуть мати один набір прав доступу, а адміністратори проміжного рівня – інші.

Будь-яка дана комбінація *securityModel* і *securityName* може належати не більше, ніж одній групі. Це означає, що для даного агента користувач, чиї зв'язки захищаються відповідно до даного значення *securityModel*, може входити тільки в одну групу.

Права доступу для групи можуть відрізнятися в залежності від ступеня захисту повідомлення, що містить запит. Наприклад, агент може дозволити доступ тільки для читання повідомлення, яке не пройшло автентифікацію, але вимагати автентифікації для одержання права запису. Далі, для особливо захищених об'єктів, агент може вимагати, щоб і запит, і відповідь передавалися з використанням сервісу таємності.

Контекст *MIB* – це іменована підмножина екземплярів об'єктів у локальній структурі *MIB*. Контексти забезпечують корисний засіб об'єднання об'єктів у сімейства з різними політиками доступу.

Поняття контексту відноситься до сфери керування доступом. Коли станція керування взаємодіє з агентом з метою одержання доступу до інформації керування агента, взаємодія здійснюється між адміністратором і процесором *SNMP* агента, а привілеї керування доступом виражаються у вигляді подання *MIB*, застосованого до цього адміністратора і даного контексту. Контексти мають такі ключові характеристики.

- Об'єкт *SNMP*, однозначно ідентифікований contextEngineID, може підтримувати кілька контекстів.
- Об'єкт або екземпляр об'єкта може з'являтися в декількох контекстах.
- Якщо існують множинні контексти, то для ідентифікації окремого екземпляра об'єкта на додаток до типу об'єкта і типу екземпляра потрібна ідентифікація значень *contextName* і *contextEngineID*.

Для певних груп буває необхідно обмежити доступ до деякої підмножини керованих об'єктів у системі агента. З цією метою доступ до контексту здійснюється за допомогою подання *MIB*, що визначає конкретну множину керованих об'єктів (а якщо необхідно, то і конкретні екземпляри об'єктів). Модель *VACM* використовує потужну і гнучку технологію визначення подань *MIB*, основану на поняттях піддерев та сімейств представлень.

Керовані об'єкти в локальній базі даних підлягають ієрархії, на основі ідентифікаторів об'єктів. Така локальна база даних охоплює підмножину всіх типів об'єктів, визначених відповідно до стандарту *Інтернет* для структур *SMI (Structure of Management Information –*

структура інформації керування), і включає екземпляри об'єктів, ідентифікатори яких відповідають угодам *SMI*.

Більш формально піддерево може бути визначене як множина всіх об'єктів і екземплярів об'єктів, що мають у своїх іменах загальний префікс ідентифікатора об'єкта мовою *ASN.1 (Abstract Syntax Notation One* – абстрактна синтаксична нотація версії 1). Найбільший загальний префікс всіх екземплярів об'єктів у піддереві є ідентифікатором об'єкта кореневої вершини цього піддерева.

З кожним елементом таблиці *vacmAccessTable* асоціюються три **подання** *MIB* – для доступу читання, запису і повідомлення. Кожне подання *MIB* складається з набору піддерев подань. Кожне піддерево подань характеризується як включене або виключене. Це значить, що будь-яке подання MIB повинне або включати, або не включати відразу всі екземпляри об'єктів, що містяться в даному піддереві. Крім того, визначається маска подання, що дозволяє зменшити обсяг конфігураційної інформації, при використовуванні тонких розмежувань прав доступу.

Модель *VACM* дає процесорові *SNMP* можливість визначити конкретний набір прав доступу, що формує **політику доступу**. На визначення прав доступу впливають такі фактори.

- **Користувач, який запитує доступ**. Модель *VACM* дозволяє агентові призначати різним користувачам різні привілеї доступу. Наприклад, керуюча система, що відповідає за конфігурацію всієї мережі, може мати широкі повноваження щодо зміни елементів локальної структури *MIB*, у той час як адміністраторові проміжного рівня із задачами моніторингу може надаватися доступ тільки для читання і тільки до деякої підмножини локальної структури *MIB*. Користувачі поєднуються в групи, яким присвоюється політики доступу.
- **Ступінь захисту**, що потрібний для запиту в повідомленні *SNMP*. Звичайно агент вимагає використання ідентифікації для повідомлень, що містять запит установлення параметрів (операцію запису).
- **Модель захисту**, використовувана для обробки повідомлення запиту. Якщо в системі агента реалізовано кілька моделей захисту, то агент може бути побудований таким чином, щоб надавати різні рівні доступу запитам. Наприклад, деякі елементи можуть бути доступними, якщо повідомлення із запитом пройшло через *USM*, і недоступними, якщо використовувалася модель захисту *SNMPv1*.
- **Контекст** *MIB* для даного запиту.
- **Конкретний екземпляр об'єкта**, для якого запитується доступ. Деякі об'єкти в порівнянні з іншими містять більш важливу або секретну інформацію, тому політика доступу може залежати від конкретного екземпляра об'єкта, зазначеного в запиті.
- **Тип доступу** (читання, запис, повідомлення), який запитується.

Читання, запис і повідомлення є різними операціями керування, і з кожної з них можуть пов'язуватися різні політики керування доступом.

5.6.5 Керування доступом

Додаток *SNMP* викликає модель *VACM* за допомогою примітива isAccessAllowed з параметрами *securityModel*, *securityName*, *securityLevel*, *viewType*, *contextName* і *variableName*. Усі ці параметри необхідні для того, щоб прийняти рішення щодо керування доступом. Іншими словами, підсистема керування доступом є досить гнучким засобом, у якій система ухвалення рішення розділяється на шість окремих частин.

На рис. 5.12 показана схема введення змінних і внутрішніх таблиць *MIB* моделі *VACM*, що беруть участь у процесі ухвалення рішення керування доступом.

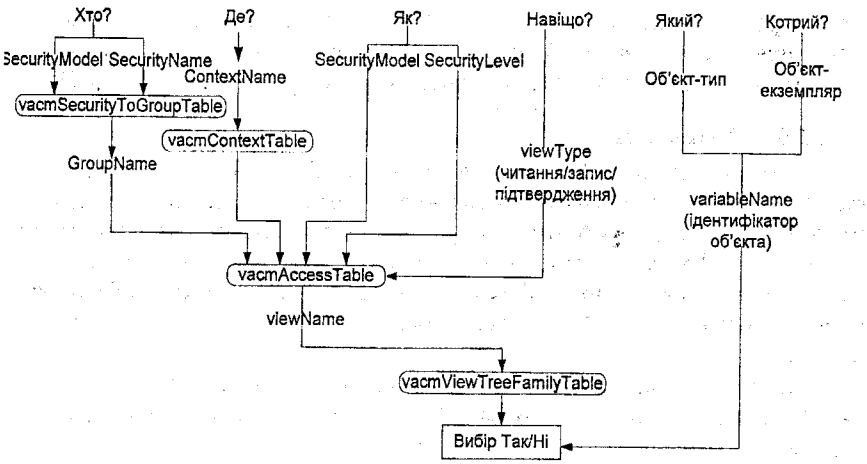


Рисунок 5.12 – Логіка VACM

- **Хто?** Комбінація *securityModel* (модель захисту) і *securityName* (ім'я захисту) відповідає на запитання: хто вимагає дану операцію. Ця комбінація ідентифікує користувача, чий зв'язок захищається даною моделлю захисту *(securityModel)*. Комбінація належить не більш, ніж одній групі в даному процесорі *SNMP*. Ім'я групи *(groupName)* визначається за допомогою таблиці *(vacmSecurityToGroupTable)* за даними значеннями *securityModel* та *securityName*.
- **Де?** Значення *contextName* (ім'я контексту) указує, де варто шукати необхідний об'єкт керування. При цьому таблиця *vacmContextTable* містить список розпізнаваних значень *contextName*.
- **Як?** Комбінація *securityModel* і *securityLevel* (ступінь захисту)

визначає, як захищається вхідний запит або *PDU* типу *Inform*. Комбінація об'єктів хто, де і як може відповідати тільки одному елементові таблиці *vacmAccessTable* (таблиця доступу) або жодному.

- **Навіщо**? Значення *viewType* (тип подання) указує, навіщо потрібен доступ: для читання, запису або повідомлення. Обраний елемент таблиці *vacmAccessTable* містить по одному значенню *viewName* (ім'я подання) *MIB* для кожного з цих трьох типів дій, і *viewType* використовується для вибору конкретного *viewName*. Це значення задає відповідне подання *MIB* з таблиці *vacmViewTreeFamilyTable*.
- **Який**? Значення *variableName* (ім'я змінної) є ідентифікатором об'єкта, префікс якого вказує тип об'єкта, а суфікс – його екземпляр. Тип об'єкта говорить про те, який тип інформації керування запитується.
- **Котрий**? Екземпляр об'єкта вказує, що з наявних елементів інформації запитується.
- Нарешті, значення *variableName* розглядається в рамках розширеного подання *MIB*. Доступ дозволяється, якщо значення *variableName* відповідає елементові, включеному в дане подання *MIB*.

5.6.6 Мотивація

Використовувані при побудові моделі *VACM* поняття призводять до досить складного визначення керування доступом. Причиною їхнього введення є необхідність повного розуміння зв'язків, використовуваних при доступі до інформації керування, а також мінімізації обчислювального навантаження і вимог до пам'яті в агенті. У *SNMPv1* поняття об'єднання використовується для того, щоб подати таку пов'язану з захистом інформацію:

- ідентифікуючий запитуючий об'єкт (станція керування);
- ідентифікуючий виконуючий об'єкт (агент, що діє від свого імені або від імені об'єкта);
- ідентифікуюче розміщення інформації керування, до якої потрібен доступ (агент або об'єкт);
- автентифікація;
- керування доступом (авторизація виконання необхідної операції);
- подання *MIB*.

При об'єднанні всіх цих понять в одній змінній губиться гнучкість і обмежуються функціональні можливості. Модель *VACM* дозволяє передати пов'язану із захистом інформацію за допомогою використання своєї окремої змінної для кожного елемента. Така модель виявляється важливим удосконаленням у порівнянні з *SNMPv1*, оскільки різні поняття виявляються розділеними, а їхні значення вибираються окремо.

6 ЗАХИСТ ПОШТОВИХ ПОВІДОМЛЕНЬ

Електронна пошта була першою, а спочатку, мабуть, єдиною мережевою службою, яку почали застосовувати користувачі обчислювальних мереж. З появою мереж загального призначення, таких, як *Інтернет*, вона почала стрімко розвиватися, набуваючи з кожним роком все більше прихильників.

В Україні відправити повідомлення в електронному вигляді може сьогодні будь-який громадянин. Проте слід мати на увазі, що засоби звичайної електронної пошти навряд чи можуть забезпечити конфіденційність переданої інформації, авторство і цілісність електронних документів. Проте недосвідченому користувачеві не варто зневірятися: у його розпорядженні є і захищена електронна пошта.

6.1 Основні проблеми електронної пошти

Більшість проблем, з якими стикаються користувачі електронної пошти (спам, віруси, різноманітні атаки на конфіденційність листів тощо), пов'язана з недостатнім захистом сучасних поштових систем.

З цими проблемами доводиться мати справу і користувачам загальнодоступних публічних систем, і організаціям. Практика показує, що одномоментне вирішення проблеми захисту електронної пошти неможливе. Спамери, творці і розповсюджувачі вірусів, хакери винахідливі, і рівень захисту електронної пошти, цілком достатній вчора, сьогодні може виявитися недостатнім. Для того, щоб захист електронної пошти був на максимально можливому рівні, а досягнення цього рівня не потребувало надмірних зусиль і витрат, необхідний систематичний і комплексний, з урахуванням всіх погроз, підхід до рішення даної проблеми.

6.1.1 Боротьба із спамом та вірусами

Сьогодні доступні багато програмних продуктів, в тому і числі і безкоштовних, призначених для боротьби з вірусами. Що ж до рішень по боротьбі із спамом, тут можливі декілька варіантів захисту.

Можна реалізувати систему фільтрів, що дозволяють відсікати вхідну кореспонденцію за адресою, темою або вмістом листа. Фільтри зазвичай розміщуються на клієнтській стороні, і користувач сам може задавати необхідні параметри. Як приклад можна назвати системи *Spam Buster*, *MailWasher*, *Active Email Monitor* (*VicMan Software*), *eMailTrackerPro* тощо. Окрім фільтрації спаму такі програми можуть виконувати функції очищення поштової скриньки, перевірки пошти, читання заголовків листів і так далі

Система фільтрів встановлюється на поштовому сервері; листи, що в такому разі нагадують спам, відсікаються ще до попадання в ящик користувача. Також може бути реалізований захист на основі «спам-

листів», що містять список *Інтернет*-провайдерів, з адрес яких проводиться несанкціоноване розсилання рекламного характеру. Прикладами можуть служити служба *Mail-Filtering Service* проекту *Mail Abuse Prevention Project* та *Realtime Blackhole List*, база даних відкритих поштових серверів (під відкритістю в даному випадку розуміється відсутність адекватного адміністрування, що призводить до неконтрольованих розсилань спаму через такі поштові сервери).

Огляди подібних продуктів регулярно публікуються. Створити систему антивірусного та антиспамового захисту загалом нескладно як для користувача загальнодоступного комунікаційного середовища, так і для служби безпеки організації, що розвернула на своїй обчислювальній інфраструктурі корпоративну поштову систему. Після вибору і установлення засобів антивірусного та антиспамового захисту найголовніше – їх акуратне і своєчасне оновлення. Головне пам'ятати, що думки зловмисників не стоять на місці, а спамери день від дня стають все активнішими.

6.1.2 Хакери

Передумови деяких проблем, пов'язаних безпосередньо з конфіденційністю поштових повідомлень, закладалися при виникненні електронної пошти три десятиліття тому. Багато в чому вони не вирішені до цих пір.

Жоден із стандартних поштових протоколів (*SMTP*, *POP3*, *IMAP4*) не включає механізмів захисту, які гарантували б конфіденційність листування.

Відсутність надійного захисту протоколів дозволяє створювати листи з фальшивими адресами. Не можна бути упевненим на 100% в тому, хто є дійсним автором листа.

Електронні листи легко змінити. Стандартний лист не містить засобів перевірки власної цілісності і при передачі через множину серверів, може бути прочитаний і змінений; електронний лист схожий сьогодні на листівку.

Зазвичай в роботі електронної пошти немає гарантій доставки листа. Не дивлячись на наявність можливості отримати повідомлення про доставку, часто це означає лише, що повідомлення дійшло до поштового сервера одержувача (але не обов'язково до самого адресата).

Як видно з опитувань, проблема захисту вмісту електронного листування від цікавості і втручання сторонніх осіб, забезпечення її конфіденційності сьогодні явно недооцінюється. Цьому є і суб'єктивне пояснення. З появою *Інтернет* народився стереотип, що до повідомлень, які пересилаються по електронній пошті, добратися важче, ніж до паперових документів. Проте дуже часто цінність даних, що відкрито пересилаються, набагато перевищує вартість доступу до них. Ніхто не

дасть гарантії, що персонал *Інтернет*-провайдерів і комунікаційних вузлів не використовує свій доступ до чужого листування в корисних (або просто непристойних) цілях, а зловмисники можуть витягувати масу конфіденційних відомостей з абсолютно некорисного, на перший погляд, тексту. Так, варто тільки конкурентам заплатити за перехоплення пошти, і робота компанії може припинитися, а потенційні партнери отримають вигідні пропозиції від іншої організації.

Які найбільш типові засоби використовують зловмисники для атак на системи електронної пошти? Це можуть бути сніфери. З огляду на те, що деякі мережеві застосування, зокрема поштові, передають дані в текстовому форматі (*SMTP*, *POP3* тощо), за допомогою сніфера можна дізнатися текст листа, імена користувачів і паролі.

Інший спосіб – *IP*-спуфінг – можливий, коли зловмисник видає себе за санкціонованого користувача. Атаки *IP*-спуфінгу часто є відправною точкою для інших атак, наприклад, *DOS*. Зазвичай *IP*-спуфінг обмежується вставкою помилкової інформації або шкідливих команд в звичайний потік передаваних по мережі даних. Це відбувається у випадку, якщо головне завдання полягає в отриманні важливого файла. Проте зловмисник, помінявши таблиці маршрутизації даних і направивши трафік на помилкову *IP*-адресу, може сприйматися системою як санкціонований користувач і, отже, мати доступ до файлів, додатків, і електронної пошти.

Атаки для отримання паролів можна проводити за допомогою цілого ряду методів, і хоча вхідне ім'я і пароль можна отримати за допомогою *IP*-спуфінгу і перехоплення пакетів, їх часто намагаються підібрати шляхом простого перебору за допомогою спеціальної програми.

Ще один тип атаки на конфіденційність – *Man-in-the-Middle* («людина в середині») – полягає в перехопленні всіх пакетів, переданих по маршруту від провайдера в будь-яку іншу частину мережі. Подібні атаки з використанням сніферів пакетів, транспортних протоколів і протоколів маршрутизації проводяться з метою перехоплення інформації, отримання доступу до приватних мережевих ресурсів, спотворення переданих даних. Вони цілком можуть використовуватися для перехоплення повідомлень електронної пошти і їх зміни, а також для перехоплення паролів і імен користувачів.

І, нарешті, атаки на рівні додатків використовують добре відомі слабкості серверного програмного забезпечення (*sendmail*, *HTTP*, *FTP*). Можна, наприклад, дістати доступ до комп'ютера від імені користувача, що працює з додатком тієї ж електронної пошти.

6.1.3 Основні методи і засоби захисту від атак на електронне листування

Для захисту мережевої інфраструктури використовується немало всіляких заслонів і фільтрів: *SSL*, *TSL*, віртуальні приватні мережі. Основні

методи захисту від атак хакерів будуються саме на основі цих засобів. Це, перш за все, сильні засоби автентифікації, наприклад, технологія двофакторної автентифікації, при якій відбувається поєднання того, що у вас є, з тим, що ви знаєте. Ця технологія використовується, наприклад, в роботі звичайного банкомату, який ідентифікує по картці і за кодом. Для автентифікації в поштовій системі теж буде необхідно використовувати «картку» – програмний або апаратний засіб, що генерує за випадковим принципом унікальний одноразовий пароль. Його перехоплення немає сенсу, оскільки він буде вже використаний і виведений з використання. Проте така міра ефективна тільки проти перехоплення паролів, але не проти перехоплення іншої інформації (наприклад, повідомлень електронної пошти).

Інші засоби захисту полягають в ефективній побудові та адмініструванні мережі. Мова йде про побудову комутованої інфраструктури, заходи контролю доступу і фільтрації вихідного трафіка, закриття «дірок» в програмному забезпеченні та регулярне оновлення, установлення антивірусних програм тощо.

І, нарешті, найефективніший метод – криптографія, яка не запобігає перехопленню інформації і не розпізнає роботу програм для цієї мети, але робить цю роботу даремною. Криптографія також захищає від *IP*-спуфінгу, якщо використовується при автентифікації. Найширше для криптографічного захисту переданих по каналах зв'язку даних, включаючи листи електронної пошти, застосовується протокол *SSL*, в якому для шифрування даних використовуються ключі *RSA*. Проте *SSL* захищає листи тільки при передачі; якщо не використовуються інші засоби криптозахисту, то листи при зберіганні в поштових скриньках і на проміжних серверах знаходяться у відкритому вигляді.

Сукупність всіх цих засобів можна зобразити як багаторівневу ешелоновану систему оборони. І, тим не менше, як показує практика, існує можливість пробратися крізь всі ці рівні та дістати доступ до даних. Отже, останнім рубежем оборони є криптографічні засоби захисту всередині системи електронної пошти.

6.1.4 «Сильні» криптоалгоритми

Найефективнішим способом захисту листів електронної пошти від перехоплення фахівці з безпеки комп'ютерних мереж визнають їх кодування на основі «сильних» криптографічних алгоритмів. Таке кодування і формування електронного підпису роблять неможливою зміну листа і дозволяють легко виявляти підроблені листи. Існує велике число алгоритмів і протоколів шифрування. Серед алгоритмів симетричної криптографії, яких велика кількість, можна згадати *RC4*, *RC5*, *CAST*, *DES*, *AES* і так далі Оптимальна довжина ключів шифрування для цих алгоритмів — 128 розрядів. Що стосується асиметричного шифрування, то

тут в основному використовуються алгоритми *RSA*, Діффі-Хелмана, Ель-Гамала, при цьому довжина ключів шифрування зазвичай складає 2048 розрядів. За оцінками дослідницьких компаній, у всьому світі лише близько 1% користувачів публічних поштових систем використовували кодування на основі криптографії, а серед користувачів корпоративних поштових систем таких не більше 10-15%.

У чому причина? Перш за все, в складності використання засобів кодування, як правило, зовнішніх стосовно програмного забезпечення електронної пошти. Навіть пакети на основі стандарту *OpenPGP*, найбільш популярні засоби кодування електронного листування, що достатньо прості у використанні, вимагають додаткових навиків і дій. Крім того, і ця причина виявляється суттєвою, відкриті ключі розподілені між учасниками листування не можуть бути рішенням для широких кругів користувачів.

Основна складність застосування такого роду систем пов'язана з необхідністю створення та супроводу *PKI* (*Personal Key Infrastructure*) – інфраструктури розподілу, зберігання і захисту ключів учасників електронного листування.

Останніми роками розроблено декілька систем електронної пошти, в яких з урахуванням ряду допущень проблема розгортання *PKI* для користувачів вирішена. По-перше, передбачається, що всі учасники даної системи електронної пошти можуть писати один одному, по-друге, процедура генерації секретного ключа відбувається на основі пароля користувача. Ці допущення дозволили повністю автоматизувати всю роботу з відкритими і секретними ключами. Немає необхідності розподіляти за користувачами відкриті ключі: вони можуть зберігатися на відкритому сервері і доступ до них може відбуватися за іменем користувача. Користувачі в явному вигляді не працюють з ключами, а виконують тільки типові операції обробки листів.

Прикладом такої системи є служба захищеної електронної веб-пошти *S-Mail.com*, у якій захист даних на основі «сильних» криптоалгоритмів (для кодування даних використовується *PGP* з ключем *RSA* завдовжки 2048 байт; у протоколі *SSL* використовується ключ завдовжки в 1024 байт) реалізований спочатку. Реалізовані заходи для запобігання небезпекам, пов'язані з використанням мови розмітки гіпертекстових документів *HTML* для написання листів.

6.1.5 Захист корпоративної поштової системи

Поштову систему із засобами криптозахисту має сенс використовувати як корпоративну поштову систему, яку можна розвернути на власній IT-інфраструктурі, проблеми безпеки при цьому розв'язуються найчастіше за рахунок встановлення шлюзу/заслонів на з'єднанні корпоративної мережі з *Інтернет* і на поштовому сервері. Цей варіант призначений, перш за все, для великих організацій із сильними IT-

підрозділами і великими бюджетами. Для середніх і малих організацій переважно варіант оренди корпоративної поштової системи у *ASP*-провайдера. Від корпоративних поштових систем часто вимагають додаткових функцій підтримки спільної діяльності співробітників компанії. Як корпоративні поштові системи часто згадуються *Lotus Notes* і *Microsoft Exchange*, які містять дуже багато цих додаткових функцій і менш підходять для веб-хостингу, чим «легкі» корпоративні поштові системи, які за функціональністю мало відрізняються або зовсім не відрізняються від загальнодоступних поштових систем.

6.2 Системи електронної пошти

Практично будь-яка програма електронної пошти, наприклад *MS Outlook*, має вбудовані засоби шифрування і електронного цифрового підпису (*ЕЦП*). Користувачеві залишається з'ясувати, чи застосовувати ці засоби захисту або скористатися альтернативними системами захищеної електронної пошти.

При виборі системи захищеної електронної пошти спочатку слід вирішити питання, яку з систем електронної пошти узяти за основу, а потім визначити програму поштового клієнта, яка і забезпечить захист переданої інформації.

Існує значне число систем електронної пошти, вони відрізняються реалізацією протоколів і форматами переданих поштових повідомлень. Проте практично всі поштові системи залежно від стандартів, на яких вони грунтуються, поділяються на три класи:

- поштові системи, основані на протоколі *SMTP*;
- поштові системи, основані на протоколі *X.400*;
- поштові системи, основані на оригінальних протоколах.

Поштова система, основана на протоколі *SMTP* – найвідоміша, оскільки де-факто вона широко використовується як стандарт для передачі повідомлень в *Інтернет*. Протокол *SMTP*, хоча і відноситься відповідно до моделі *OSI* до прикладного рівня, взаємодіє тільки з протоколами транспортного рівня.

Сучасний протокол *SMTP* не залежить від транспортного середовища і може використовуватися для доставки пошти в мережах з протоколами, відмінними від *TCP/IP* та *X.25*. Це досягається за рахунок концепції *IPCE* (*InterProcess Communication Environment*), яка дозволяє взаємодіяти процесам, які підтримують *SMTP* в інтерактивному режимі.

Поштова система на основі протоколу *X.400* на відміну від системи *SMTP* спочатку базувалася на рекомендаціях, розроблених Міжнародним Консультативним Комітетом з Телефонії і Телеграфії (зараз Сектор телекомунікацій Міжнародного союзу електрозв'язку). Рекомендації *X.400* охоплюють всі сторони побудови і функціонування системи електронної пошти, включаючи склад і елементи поштової системи, протоколи

управління і передачі повідомлень, їх формати і багато іншого.

Поштові системи на основі оригінальних протоколів майже невідомі широкому колу користувачів мережі *Інтернет*. Вони довгий час знаходилися в стороні від загальної спрямованості розвитку систем електронної пошти, хоча останнім часом виявилася явна їх конвергенція з системами типу *SMTP*.

Найбільш відомими поштовими системами на основі оригінальних протоколів є *MS Mail for PC Networks* і *Lotus cc:Mail*. Ці системи основані на файловому методі доступу до поштових скриньок, мають власні стандарти форматів поштових повідомлень і протоколів їх передачі.

Поштові системи *SMTP* або *X.400* достатньо легко можна перетворити на системи захищеної електронної пошти, використовуючи для цього спеціальні протоколи криптографічного перетворення повідомлень і відповідні поштові клієнти, що їх реалізовують.

У поштових системах, основаних на протоколі *SMTP*, найбільшого поширення набули криптографічні протоколи стандарту *IETF S/MIME* (*Secure MIME*) компанії *RSA Data Security* (www.rsasecurity.com) та *PGP/MIME* (*Pretty Good Privacy MIME*) компанії *PGP Inc.* (www.pgp.com). Остання версія протоколу *PGP* носить назву *OPENPGP*. За останні роки ці стандарти істотно розширили свої функціональні можливості і зближувалися так, що тепер вони головним чином відрізняються використовуваними алгоритмами шифрування та електронного цифрового підпису і особливостями реалізації підтримки цифрових сертифікатів за стандартом *X.509*.

Відповідно до рекомендацій *RFC 1847* стандарти *PGP/MIME* і *S/MIME* визначають інкапсуляцію зашифрованого повідомлення та електронного цифрового підпису за допомогою спеціальних блоків даних (*multipart/signed* і *multipart/encrypted*) в тіло поштового повідомлення. Для обробки отриманих повідомлень в їх заголовок включається змінна *protocol*, що встановлює алгоритми шифрування та електронного цифрового підпису. Обидва стандарти припускають, що початкове повідомлення подане в 7-бітових символах, для чого використовуються алгоритми кодування *Base-64 (Radix-64)* і *Quoted printable*. Потрібно відзначити, що останні версії поштових програм реалізовують не тільки *S/MIME* (*RFC-2633*), але і розширення цієї специфікації (*RFC 2634*), що включають, наприклад, такий сервіс, як достовірне підтвердження отримання повідомлення.

Фахівці-криптографи при оцінці *PGP/MIME* і *S/MIME* сходяться на думці, що за стійкістю шифрування, тобто здатністю протистояти дешифруванню або порушенню цілісності і авторства повідомлень та електронного цифрового підпису, ці стандарти практично рівноцінні, хоча і реалізовують різні криптографічні алгоритми. Відмінності стосуються в основному відкритості використаних криптографічних алгоритмів і

особливостей підтримки цифрових сертифікатів відкритих ключів шифрування та електронного цифрового підпису.

Ряд криптографічних алгоритмів стандарту *S/MIME*, розроблених *RSA Data Security*, були запатентовані і відкрито не опубліковані. Природно, що це негативно позначилося на можливості вбудовування криптоядра в поштові клієнти, хоча компанія і розробила *Application Programming Interface* (*API*) до функцій криптографічного перетворення і достатньо докладні їх описи. Закінчення терміну дії патентів може послужити сигналом до відкритої публікації криптографічних алгоритмів стандарту *S/MIME*.

Що стосується криптографічних алгоритмів компанії PGP, всі вони публікуються відкрито. Це дозволяє широкій громадськості обговорювати їх, а криптограми – піддавати аналізу. Тому ці алгоритми викликають більшу довіру у користувачів, чим розроблені компанією *RSA Data Security*.

Специфікації відкритих ключів шифрування та електронного цифрового підпису *X.509* підтримує стандарт *S/MIME*. Сертифікат *X.509* засвідчує, що відкриті криптографічні ключі дійсно належать тому, від чийого імені були опубліковані або передані. Слід враховувати, що стандарт *S/MIME* всього лише визначає формат поштового повідомлення і не дає жодної інформації про способи отримання і перевірки цифрових сертифікатів. Ці функції реалізуються поштовими клієнтами. Стандарт *PGP/MIME* спочатку передбачав підтримку цифрових сертифікатів компанії *PGP*, які несумісні із сертифікатами *X.509*. Це створює додаткові труднощі при розробці поштових клієнтів, оскільки більшість виробників мережевого програмного забезпечення орієнтована на використання інфраструктури відкритих криптографічних ключів (*PKI*). Проте останні версії стандарту *PGP/MIME* вже передбачають підтримку цифрових сертифікатів *X.509*.

6.3 Захищена електронна пошта *S/MIME*

Специфікація *Secure Multipurpose Internet Mail Extension* (*S/MIME*) призначена для захисту найбільш популярного *Інтернет*-сервісу – електронної пошти. Через важливість цього мережевого сервісу робилися багато спроб стандартизувати рішення захищеної електронної пошти. Одна з перших схем захисту – *Privacy Enhanced Mail* – була запропонована в 1985 році. Наступна схема захисту з'явилася в 1995 році і отримала назву *MIME Object Security Services* (*MOSS*). Не дивлячись на те, що в них містилася безліч хороших ідей, негнучкість і відсутність сумісності відсунули *PEM* і *MOSS* на другі ролі у сфері створення захищеного обміну повідомленнями і були витиснені системами *Secure/MIME* (*S/MIME*) і *Pretty Good Privacy* (*PGP*).

Протоколи *S/MIME* і *PGP* підтримуються більшістю програмних

продуктів, призначених для колективної роботи і захисту електронної пошти. Найбільш поширеним і широковизнаним стандартом обміну повідомленнями став розроблений в 1996 році стандарт *S/MIME*. Спочатку компанією *RSA Data Security* була запропонована друга версія специфікації *S/MIME v2*, пізніше вона була доопрацьована робочою групою організації *IETF* і в результаті з'явилася нова, третя версія – *S/MIME v3*.

S/MIME забезпечує захист від трьох типів порушень безпеки: "підслуховування", або перлюстрації, спотворення повідомлень і фальсифікації. Захист від перлюстрації досягається шляхом шифрування повідомлень. Для захисту від спотворення поштового повідомлення або фальсифікації в *S/MIME* застосовують цифрові підписи. Цифровий підпис гарантує, що повідомлення не було змінене в процесі передачі. Крім того, його наявність не дозволяє відправникові повідомлення відмовитися від свого авторства.

Проте цифровий підпис не гарантує конфіденційність повідомлення. У *S/MIME* цю функцію виконує цифровий конверт. Шифрування здійснюється за допомогою симетричного криптографічного алгоритму типу *DES*, потрійний *DES* або *RS2*. Симетричний ключ шифрується за допомогою відкритого ключа одержувача, а зашифроване повідомлення і ключ передаються разом.

Специфікація *S/MIME* визначає два типи *MIME*-конверта: один для цифрових підписів, інший – для шифрованих повідомлень. Обидва типи базуються на синтаксисі криптографічних повідомлень стандарту *PKCS#7*. Якщо повідомлення повинне бути зашифроване, а шифртексту повинні бути присвоєні деякі атрибути, використовуються вкладені конверти. Зовнішній і внутрішній конверт призначаються для захисту цифрових підписів, а проміжний конверт – для захисту шифртекста.

Окрім забезпечення цілісності повідомлення під час передачі, *S/MIME* ідентифікує власника конкретного відкритого ключа за допомогою сертифіката *X.509*. Цифровий сертифікат засвідчує, що відкритий ключ дійсно належить тому, хто є суб'єктом сертифіката.

Протокол *S/MIME v3* підтримує такі важливі властивості, які відсутні в *S/MIME v2*: завірені цифровим підписом квитанції; мітки безпеки; списки розсилки; гнучке управління ключами.

Завірені цифровим підписом квитанції дозволяють відправникові повідомлення упевнитися в тому, що воно було отримане адресатами без змін. Одержувач повідомлення не може згенерувати валідну квитанцію до тих пір, поки не перевірить підпис відправника на отриманому повідомленні.

Мітки безпеки дозволяють відправникові задавати вимоги до змісту повідомлення. Найчастіше мітка безпеки свідчить про включення в зміст повідомлення приватної або конфіденційної інформації.

Зазвичай відправник повідомлення шифрує його один раз за

допомогою симетричного ключа. Потім відправник повинен зашифрувати симетричний ключ окремо, використовуючи відкритий ключ того адресата, якому він направляє своє повідомлення. Якщо кількість адресатів велика, виникає необхідність делегувати функції шифрування довіреному серверу. Такий сервер називають агентом списку розсилки (*Mail List Agent – MLA*). Якщо є агент *MLA*, то відправник може послати зашифроване повідомлення йому, а той, у свою чергу, після виконання відповідних операцій виконує розсилання повідомлення всім адресатам із списку відправника. При цьому *MLA* не дістає доступу до ключа шифрування повідомлення і не розшифровує повідомлення, що пересилається.

Якщо адресати повідомлення територіально розподілені (наприклад, знаходяться на різних континентах), то відправник посилає зашифроване повідомлення головному агенту *MLA*, головний агент пересилає його регіональним агентам *MLA*, і, нарешті, кожен регіональний агент доставляє зашифроване повідомлення локальним одержувачам. В цьому випадку повідомлення бере участь в кожній міжконтинентальній комунікації тільки один раз. Правда, якщо в списки розсилки кожного регіонального *MLA* включені адреси всіх інших регіональних агентів, може відбуватися багатократне пересилання повідомлення по коду. Для виявлення циклів аналізується атрибут *історія розповсюдження списку розсилки*, що міститься в зовнішньому конверті повідомлення. Коли агент *MLA* отримує повідомлення, то перевіряє історію розповсюдження, щоб визначити, чи не оброблялося вже дане повідомлення. Якщо повідомлення оброблялося, воно просто віддаляється. Зовнішній конверт з цифровим підписом забезпечує цілісність історії розповсюдження списку розсилки.

S/MIMEv2 забезпечує тільки транспортування ключів шифрування повідомлень, а *S/MIMEv3* додатково підтримує механізми узгодження і зовнішнього розповсюдження симетричних ключів шифрування.

Всі сервіси, запропоновані *S/MIME* (обох версій), покладаються на сертифікати і надійність скріплення електронної адреси суб'єкта з його відкритим ключем. Адреса електронної пошти (часто звана адресою *RFC 822*) повинна бути подана в доповненні сертифіката *Subject Alternative Name* (альтернативне ім'я суб'єкта). Якщо використовується друга версія *S/MIME*, то адреса електронної пошти указується як ім'я суб'єкта (*emailAddress*).

Відправник зашифрованого повідомлення повинен бути упевнений, що відкритий ключ належить саме тому одержувачеві, якому він адресує своє повідомлення, інакше доступ до вмісту повідомлення може отримати сторонній. Аналогічно, розповсюджуючи ключі шифрування симетричного ключа тільки членам списку розсилки відправника, агент *MLA* покладається на правильність зв'язування ідентичності одержувача і його відкритого ключа. Відправник і агент *MLA* порівнюють ідентифікаційну ознаку одержувача, вказану в сертифікаті, з адресою

електронної пошти одержувача, якому посилає своє повідомлення відправник.

Одержувач повідомлення, завіреного цифровим підписом, повинен бути упевнений, що відкритий ключ підпису, який необхідний для верифікації підпису на повідомленні, належить відправникові. Для цього він порівнює адресу електронної пошти, вказану в полі *SENDER* (або *FROM*) отриманого повідомлення, з адресою, поданою в сертифікаті.

Аналогічні дії виконуються при перевірці квитанцій, завірених цифровим підписом, – для цього перевіряючий порівнює адресу електронної пошти із запиту на квитанцію з адресою електронної пошти в сертифікаті особи, що підписала квитанцію.

6.4 Сертифікати *PGP*

Система *PGP* (*Pretty Good Privacy*) розроблена американським програмістом Філіпом Циммерманном для захисту секретності файлів і повідомлень електронної пошти в глобальних обчислювальних і комунікаційних середовищах. Ф. Циммерманн запропонував першу версію *PGP* на початку 1990-х років. Версія *2.x PGP* була опублікована декількома роками пізніше в специфікації набору стандартів *IETF*, названою *PGP Message Exchange Formats*. Остання версія *PGP*, що отримала назву *Open PGP*, була видана в специфікації набору стандартів *IETF – Open PGP Message Format*. Документ, що відноситься до *Інтернет*-стандартів, об'єднує *PGP* і *MIME* і називається *PGP MIME Security with Pretty Good Privacy*.

PGP є гібридною системою, яка комплексно використовує переваги асиметричних і симетричних криптографічних алгоритмів. З погляду користувача, *PGP* поводиться як система з відкритим ключем. Вона забезпечує безпечний обмін повідомленнями і файлами по каналах відкритого зв'язку без наявності захищеного каналу для обміну ключами. *PGP* дозволяє шифрувати, завіряти електронним цифровим підписом, розшифровувати і перевіряти повідомлення під час відправки і читання електронної пошти. У *PGP* застосовуються стійкі криптографічні алгоритми *CAST*, потрійний *DES* і *IDEA*. Для вироблення сеансового ключа використовуються алгоритми *RSA* і Діффі-Хелмана, для підпису – *RSA* і *DSA*. *PGP* задає формати пакетів, що дозволяють пересилати від одного суб'єкта до іншого повідомлення і файли, а також *PGP*-ключі (*PGP*-сертифікати).

Перед роботою в системі *PGP* користувачеві необхідно згенерувати відкритий і секретний ключі. Відкритий ключ може бути переданий абонентові як повідомлення електронної пошти або як файл поміщений на сервер відкритих ключів. Отримавши копію будь-якого відкритого ключа, користувач може додати його на свою зв'язку відкритих ключів. Переконатися в тому, що ключ не був підроблений, можна, порівнюючи

унікальний відбиток своєї копії ключа з відбитком оригінальної копії. Відбиток – це рядок з цифр і букв, що ідентифікує власника ключа. Після перевірки валідності ключа користувач підписує його, підтверджуючи системі *PGP* безпеку його використання. При цьому користувач може вказати ступінь довіри до власника ключа, в сенсі його здатності ручатися за достовірність ключів третіх осіб. Ступінь довіри до здатності власника ключа виступати як посередник відображає оцінку не тільки його персональної порядності, але і компетентності в розумінні механізму управління ключами і схильності керуватися здоровим глуздом при ухваленні рішення про сертифікацію ключа третьої особи. Користувач може позначити особу як таку, що користується повною довірою, обмеженою довірою або що не користується довірою. Ця інформація про ступінь довіри до конкретного власника ключа зберігається на низці разом з відповідними ключами, але при експорті ключа із низки вона не передається, оскільки вважається конфіденційною.

При оцінці валідності відкритого ключа в системі *PGP* перевіряється рівень довіри, наданий користувачем всім підписам, якими він сертифікований. Система обчислює зважене значення валідності, при цьому два підписи осіб, що користуються обмеженою довірою, прирівнюються до підпису однієї особи, що користується повною довірою. Ключі, сертифіковані посередниками, яким довіряє користувач, *PGP* вважає валідними. Ключі, що належать цим посередникам, самі повинні бути сертифіковані користувачем або іншими посередниками, яким користувач довіряє. Таким чином, формується мережа поручительства учасників системи *PGP* за достовірність поширюваних ключів, так звана мережа довіри. Надання всім користувачам можливості діяти як посередники вважається за доцільне для децентралізованих середовищ.

PGP підтримує централізований сценарій, відповідно до якого сертифікати відкритих ключів користувачів завіряє своїм підписом особа, яка користується загальною довірою. Будь-якому відкритому ключу, завіреному таким підписом, можна довіряти в тому сенсі, що він належить тому, чиє ім'я він несе. Всі користувачі повинні володіти копією такого відкритого ключа для перевірки його цифрового підпису. *PGP* забезпечує інтегровану підтримку розповсюдження і пошуку відкритих ключів на серверах ключів. Деякі організаційні середовища використовують ієрархію засвідчуючих центрів, яка лежить в основі стандартної схеми, основаної на централізованому контролі і примусово централізованій довірі. Ієрархія засвідчуючих центрів зазвичай диктує користувачеві, кому він повинен довіряти. Децентралізований імовірнісний метод визначення валідності ключів, реалізований *PGP*, дозволяє користувачеві самостійно ухвалювати рішення про довіру, будуючи свою власну піраміду сертифікації.

Між *PGP*-ключами (або сертифікатами) і сертифікатами відкритих ключів *X.509*, а також між відповідними моделями довіри є істотні

відмінності, що перешкоджають взаємодії співтовариства користувачів *PGP* із з іншими співтовариствами, які використовують сертифікати формату *X.509* (наприклад, співтовариством користувачів *S/MIME*). Це набагато серйозніше, ніж проблема несумісності протоколів, тому що несхожа і несумісна основа базових сервісів безпеки, що забезпечуються відкритими ключами. Можливим рішенням може бути адаптація сертифікатів *X.509v3* на додаток до *PGP*-сертифіката (або замість нього). Версія *OpenPGP 6.5.* здатна підтримувати сертифікати *X.509*, але, дозволяючи користувачам *OpenPGP* підключатися до *PKI* на базі *X.509*, вона не вирішує проблему несумісності основних протоколів *OpenPGP* і *S/MIME*. Іншим можливим рішенням може бути розробка продуктів, що підтримують сертифікати і *PGP*, і *X.509 v3*, але це через істотні відмінності в моделях довіри ускладнить адміністрування і управління.

6.5 Сервіс *PGP*

Сервіс *PGP* складається з п'яти функцій: автентифікація, конфіденційність, стиснення, сумісність на рівні електронної пошти і сегментація, а також окрема функція управління ключами.

Характеристика функцій *PGP* наведена в таблиці 6.1.

Таблиця 6.1 – Коротка характеристика функцій *PGP*

Функція	Використовувані алгоритми	Опис
Цифровий підпис	*DSS/SHA* або *RSA/SHA*	З допомогою *SHA–1* створюється *хеш*-код повідомлення. Одержаний таким чином профіль повідомлення шифрується за допомогою *DSS* або *RSA* з використанням особистого ключа відправника і включається в повідомлення
Шифрування повідомлення	*CAST* або *IDEA*, або потрійний *DES* з трьома ключами і алгоритмом Діффі-Хелмана або *RSA*.	Повідомлення шифрується з допомогою *CAST-128* або *IDEA*, або *3DES* з одноразовим сеансовим ключем. Сеансовий ключ шифрується за допомогою алгоритму Діффі-Хелмана або *RSA* з використанням відкритого ключа одержувача і включається в повідомлення
Стиснення	*ZIP*	Повідомлення можна стиснути для зберігання або передачі, використовуючи формат *zip*
Сумісність на рівні електронної пошти	Перетворення у формат *radix-64*	Щоб забезпечити прозорість для всіх застосувань електронної пошти, шифроване повідомлення можна перетворити на рядок *ASCII*, використовуючи перетворення у формат *radix-64*
Сегментація	_____	Щоб задовольнити обмеження максимального розміру повідомлень, *PGP* виконує сегментацію і зворотну збірку повідомлення

Алгоритм автентифікації можна описати таким чином.

1. Відправник створює повідомлення.
2. Використовується алгоритм *SHA-1*, внаслідок чого отримується 160-бітовий хеш-вектор повідомлення.
3. Одержаний хеш-вектор шифрується за допомогою алгоритму *RSA* з використанням особистого ключа відправника, і результат додається до початку повідомлення.
4. Одержувач використовує *RSA* з відкритим ключем відправника, щоб дешифрувати і відновити хеш-код.
5. Одержувач генерує новий хеш-код одержаного повідомлення і порівнює його з дешифрованим хеш-кодом. Якщо хеш-коди збігаються, повідомлення вважається справжнім.

На рис 6.1 показана схема автентифікації повідомлення, яка використовується в сервісі *PGP*.

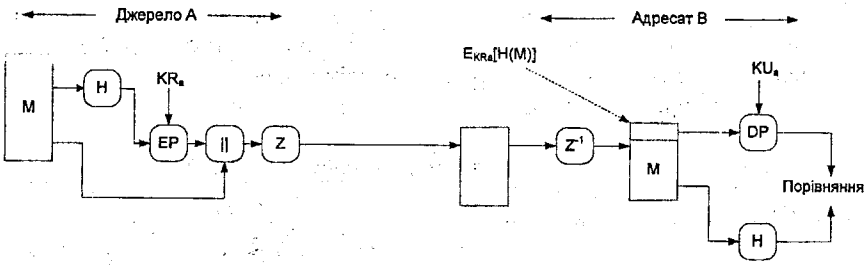


Рисунок 6.1 – Схема автентифікації

Позначення:

H – сеансовий ключ, який використовується в схемі традиційного шифрування;

KR_a – особистий ключ *A*, який використовується в схемі шифрування з відкритим ключем;

KU_a – відкритий ключ *A*, який використовується в схемі шифрування з відкритим ключем;

EP – шифрування в схемі з відкритим ключем;

DP – дешифрування в схемі з відкритим ключем;

EC – шифрування в схемі традиційного шифрування;

DC – дешифрування в схемі традиційного шифрування;

H – функція хешування;

|| – конкатенація;

Z – стиснення за допомогою алгоритму *zip*;

R64 – перетворення у формат *radix-64 ASCII*.

Алгоритм шифрування повідомлення (показаний на рис 6.2) можна описати таким чином.

1. Відправник генерує повідомлення і випадкове 128-бітове число, яке виступає як сеансовий ключ тільки для цього повідомлення.
2. Повідомлення шифрується за допомогою алгоритму *CAST-128* (можуть використовуватись алгоритми *IDEA* або *3DES*) і даного сеансового ключа.
3. Сеансовий ключ шифрується за допомогою алгоритму *RSA* і відкритого ключа одержувача і додається до початку повідомлення.
4. Одержувач використовує *RSA* з особистим ключем, щоб дешифрувати і тим самим відновити сеансовий ключ.
5. Сеансовий ключ застосовується для дешифрування повідомлення.

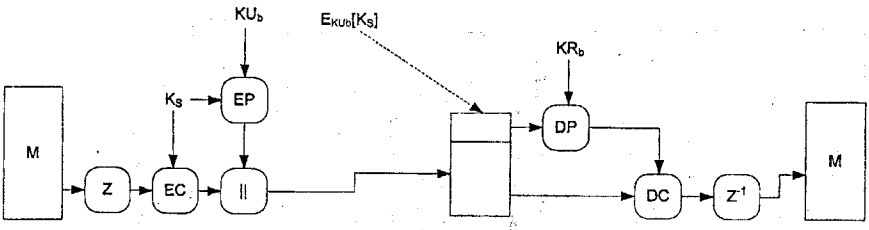


Рисунок 6.2 – Схема шифрування повідомлення.

Використання автентифікації та шифрування відбувається у такий спосіб (рис. 6.3).

Для відправника повідомлення.

1. Для повідомлення генерується підпис (хеш-вектор, зашифрований особистим ключем відправника, об'єднується з відкритим текстом повідомлення).
2. Підпис і відкритий текст повідомлення стискаються *zip*-ом.
3. Стислий відкритий текст повідомлення і підпис шифруються за допомогою алгоритму *CAST-128* (або *IDEA*, або *3DES*), а сеансовий ключ шифрується за допомогою RSA (або алгоритму Ель-Гамаля) при цьому використовується відкритий ключ одержувача.

Для одержувача повідомлення.

1. Сеансовый ключ дешифрується за допомогою особистого ключа одержувача.
2. За допомогою одержаного сеансового ключа дешифрує повідомлення.
3. Розпаковування повідомлення.
4. Відкритим ключем відправника дешифрує хеш-вектор і генерує новий хеш-вектор.
5. Порівнює їх. Якщо збігаються, то повідомлення не було змінено.

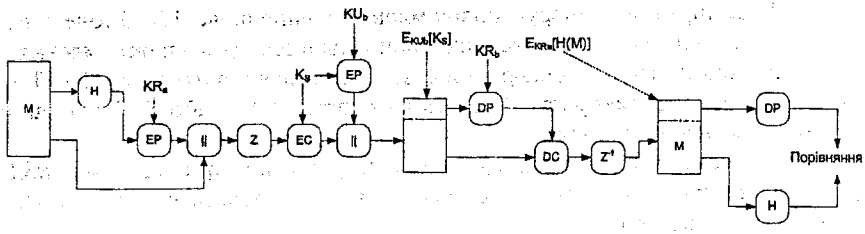


Рисунок 6.3 – Схема використання обох служб

Таблиця 6.2 – Формат повідомлення, що передається

Повідомлення			Підпис				Компонент сеансового ключа		Вміст
Дані	Мітка дати-часу	Ім'я файла	Профіль повідомлення	Перші два октети профілю повідомлення	Ідентифікатор відкритого ключа відправника (KU_a)	Мітка дати-часу	Сеансовий ключ (K_s)	Ідентифікатор відкритого ключа одержувача (RU_b)	
			E_kR_a				E_kU_a		
ZIP									
E_{Ks}									
R64									

6.6 Ідентифікатори ключів

Оскільки одержувач повідомлення має можливість отримувати зашифровані і підписані повідомлення від багатьох учасників листування, то він повинен мати декілька пар особистих відкритих ключів. Для того, щоб одержувачу визначити, який особистий ключ (алгоритм *RSA*) треба використовувати для розшифровки сеансового ключа (алгоритм *CAST-128*), він одержує ідентифікатор відкритого ключа (замість ключа пересилається його ідентифікатор, оскільки відкритий ключ для *RSA* може мати довжину в сотні десяткових розрядів). Ідентифікатор, що пов'язується з кожним відкритим ключем, розміщується в молодших 64 розрядах ключа.

Ідентифікатор ключа потрібен і для цифрового підпису *PGP*. Через те, що відправник може скористатися одним з декількох особистих ключів для шифрування профілю повідомлення, одержувач повинен знати, який відкритий ключ йому слід використовувати. Тому поле цифрового підпису повідомлення включає 64-бітовий ідентифікатор відповідного відкритого ключа. При отриманні повідомлення одержувач перевіряє, чи ідентифікатор відповідає відомому йому відкритому ключу відправника, а

потім продовжує перевірку підпису.

Компонент підпису включає такі елементи.

Мітка дати-часу. Час створення підпису

Профіль повідомлення. 160-бітний профіль повідомлення створюється за допомогою *SHA-1* і зашифровується з використанням особистого ключа підпису відправника (KR_a). Профіль обчислюється для мітки дати-часу підпису, пов'язаною конкатенацією з даними компонента повідомлення. Включення мітки дати-часу підпису в профіль забезпечує захист від атак відтворення повідомлення. Виключення імені файла і мітки дати-часу компонента повідомлення гарантує, що відокремлений підпис буде в точності збігатися з підписом, що додається в префікс повідомлення. Відокремлені підписи обчислюються для файла, в якому немає ніяких полів заголовка повідомлення.

Перші два октети профілю повідомлення. З метою автентифікації проводиться порівняння цих двох октетів відкритого тексту початкового профілю з першими двома октетами дешифрованого профілю. Ці октети також використовуються для перевірки повідомлення.

Ідентифікатор відкритого ключа відправника. Ідентифікує відкритий ключ, який використовується для дешифрування профілю повідомлення і, отже, ідентифікує особистий ключ, що використовувався для шифрування профілю повідомлення.

Компонент повідомлення і необов'язковий компонент підпису можуть бути ущільнені за допомогою *ZIP* і зашифровані з використанням сеансового ключа.

Компонент сеансового ключа включає сеансовий ключ і ідентифікатор відкритого ключа одержувача, який використовувався відправником для шифрування даного сеансового ключа.

Весь блок звичайно переводитися у формат *radix-64*. Перетворення у формат *radix-64* використовується для сумісності на рівні програмних додатків електронної пошти. Сервіс автентифікації припускає, що шифрується тільки профіль повідомлення (цифровий підпис), сервіс конфіденційності припускає, що шифрується саме повідомлення (сеансовим ключем) і підпис (за наявності останнього), таким чином частина або весь вихідний блок повідомлення є потоком довільних 8-бітових байтів. Проте багато систем електронної пошти дозволяють використовувати тільки блоки, що складаються з символів тексту *ASCII*. Щоб задовольнити таке обмеження, *PGP* забезпечує сервіс конвертації сирого 8-бітового двійкового потоку в потік друкованих символів *ASCII*. Для цього використовується схема конвертації *radix-64*.

7 ЗАХИСТ ВІД ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У загальному випадку за виявлення присутності вірусів на комп'ютері повинні відповідати антивіруси – спеціальні програми, здатні швидко і ефективно не тільки виявляти, але і знешкоджувати шкідливі програми. Проте відомо, і тому є об'єктивні причини, що жоден антивірус не забезпечує повний захист від всіх шкідливих програм. Отже, можливе зараження комп'ютера, навіть якщо на ньому встановлений антивірус. За відсутності антивірусу, вірогідність проникнення на комп'ютер шкідливих програм зростає на декілька порядків.

Якщо комп'ютер заражений невідомим вірусом, звичайною практикою є самостійне виявлення підозрілих файлів і або відправка їх на дослідження в антивірусну компанію, або самостійне знищення.

Але щоб серйозно зайнятися пошуком, потрібно мати підстави для підозр в тому, що комп'ютер заражений. Для цього потрібно знати, які особливості функціонування комп'ютера можуть бути проявами шкідливих програм.

7.1 Види проявів

Прояви шкідливих програм можна умовно розбити на три групи відповідно до того, наскільки легко їх виявити:

- **явні** – шкідлива програма самостійно проявляє помітну активність;
- **непрямі** – інші програми починають виводити повідомлення про помилки або поводитися нестандартно через присутності на комп'ютері вірусу;
- **приховані** – ні явних, ні непрямих проявів шкідлива програма не має.

7.1.1 Явні прояви

Характерні для троянських і рекламних програм. Це і зрозуміло, оскільки основною ознакою вірусів і черв'яків є здатність до зараження, для реалізації якої необхідний час. Якщо мережевий черв'як при проникненні на комп'ютер відразу ж себе виявить, користувач зможе відключити комп'ютер від мережі, перешкодивши подальшому розповсюдженню шкідливої програми.

Навпаки, троянські програми пишуться для виконання якоїсь конкретної шкідливої функції і скритність їм потрібна більшою мірою на етапі проникнення. Втім все залежить від типу трояна.

З рекламними модулями все зовсім просто: їх основна мета – залучення уваги до об'єкта реклами (веб-сайта, програмі тощо), а привернути увагу, означає виявити свою присутність.

В даний час явні прояви, як правило, так або інакше пов'язані з мережею *Інтернет*. До основних явних ознак можна віднести:

Зміна налаштувань браузера – зміна стартової сторінки браузера, зміна стандартної сторінки пошуку, несанкціоноване відкриття нових

вікон – все це може бути наслідком присутності в системі шкідливої програми. Іноді до аналогічних ефектів може призводити виконання шкідливого скрипта на одному з відвіданих сайтів. У такому разі нові програми на комп'ютер не проникають, а налаштування браузера можна відновити і повністю вирішити проблему. Якщо ж після відновлення налаштувань вони знову змінюються при запуску браузера або після перезавантаження комп'ютера, то причина змін – наявність на комп'ютері шкідливої програми. Подібна поведінка характерна для рекламних модулів, яка примусово відправляє користувачів на сайт, що рекламує якусь-небудь продукцію. А також для троянських програм, які направляють користувача на сайти, що містять інші шкідливі програми.

Спливаючі повідомлення – після установлення в системі троянська або рекламна програма виводить на екран повідомлення про те, що на комп'ютері виявлені шкідливі або рекламні програми. Такі повідомлення зазвичай зроблені схожими на стандартні службові повідомлення *MS Windows* і забезпечені гіперпосиланнями або кнопками для переходу на веб-сайт, з якого нібито можна завантажувати програму для виявлення і видалення небажаних модулів. Не дивлячись на те, що прояви достатньо явні – повідомлення на екрані, через їх маскування під службові повідомлення, користувач не завжди здогадується, що це результат роботи шкідливих програм і в результаті потрапляє на ті ж рекламні або шкідливі сайти.

Несанкціонований дозвон в Інтернет – не так давно набули поширень особливі шкідливі програми – утиліти дозвону. Ці утиліти без санкції користувача і ігноруючи налаштування намагаються встановити з'єднання з *Інтернет* через дорогу телефонну лінію або дорогого провайдера. В результаті власник комп'ютера отримує рахунок на значну суму. Отже, ознакою зараження може бути несанкціоновані спроби комп'ютера з'єднатися з *Інтернет*.

7.1.2 Непрямі прояви

На відміну від явних проявів, непрямі прояви не завжди є навмисними і нерідко викликані помилками, допущеними автором шкідливої програми.

Блокування антивірусу – зазвичай шкідлива програма проникає на захищений антивірусом комп'ютер або якщо антивірус був відключений, або якщо це порівняно нова шкідлива програма, для якої не було запису в антивірусній базі. Зрозуміло, що незабаром антивірус буде включений, або вірус буде внесений до антивірусної бази, і антивірус зможе його виявити і знешкоджувати. Щоб перешкодити цьому багато шкідливих програм небезуспішно намагаються вивантажити антивірус з пам'яті або навіть видалити файли антивірусу з дисків комп'ютера. Тому раптове завершення роботи антивірусу цілком може бути ознакою зараження.

Блокування антивірусних сайтів – оскільки вивантаження або видалення антивірусу все ж таки достатньо помітні, деякі шкідливі програми нейтралізують тільки можливість оновлення антивірусних засобів. Якщо антивірусна база не оновлюватиметься, антивірус не зможе виявляти нові віруси і стане неефективним. При цьому шкідливі програми не блокують доступ в *Інтернет* цілком – це було б дуже помітно, а тільки доступ до сайтів і серверів оновлень найбільш відомих компаній – виробників антивірусів. В середньому користувачі не часто заходять на сайти антивірусних компаній, а повідомлення антивірусу про неможливість оновитися можуть списувати на проблеми у провайдера або на самих серверах оновлення.

Збої в системі або в роботі інших програм – дуже часто причиною збоїв в роботі програм користувачі вважають присутність на комп'ютері вірусів. І хоча більшість подібних випадків на перевірку виявляються помилковою тривогою, віруси дійсно іноді можуть бути причиною збоїв. Наприклад, результатом присутності троянської програми *Backdoor.NTHack* може бути повідомлення про помилку, що виникає при завантаженні комп'ютера:

```
STOP 0x0000001e KMODE_EXCEPTION_NOT_HANDLED in win32k.sys
```

Поштові повідомлення – якщо комп'ютер заражений і розсилає інфіковані поштові повідомлення, вони можуть бути виявлені на одному з серверів в *Інтернеті* і антивірус на сервері може відправити повідомлення відправникові зараженого повідомлення. Отже, непрямою ознакою присутності вірусу може бути отримання поштового повідомлення про те, що з поштової адреси користувача комп'ютера був відправлений вірус. Проте останнім часом багато вірусів підміняють адресу відправника і отримання описаного повідомлення не обов'язково означає, що комп'ютер заражений. Через те, що формальна адреса відправника, вказана в поштовому повідомленні, може не мати ніякого відношення до зараженого комп'ютера, антивірусні програми часто взагалі не посилають повідомлень відправникам заражених повідомлень.

7.1.3 Приховані прояви

У відсутність явних або непрямих проявів про присутність вірусу можна судити, наприклад, за незвичайною мережевою активністю, коли жодне мережеве застосування не запущено, а значок мережевого з'єднання сигналізує про обмін даними. Іншими ознаками можуть служити незнайомі процеси в пам'яті або файли на диску.

Проте в наш час на комп'ютерах зазвичай встановлено так багато різних програм, що більшість файлів і процесів невідомі звичайному користувачеві. В той же час пошук прихованих проявів – це вже фактично пошук тих самих підозрілих файлів, які потрібно відправити на аналіз в антивірусну компанію.

7.2 Де шукати ознаки шкідливого програмного забезпечення

Як видно, ні непрямі, ні навіть явні прояви не можуть служити підставою для упевненості в тому, що комп'ютер заражений. Завжди існує ймовірність, що спостережуваний ефект не є результатом дій вірусу, а викликаний звичайними помилками у використовуваних програмах або ж шкідливими скриптами, які не залишили ніяких файлів на комп'ютері.

Для того, щоб підозри переросли в упевненість потрібно провести додатковий пошук прихованих проявів шкідливих програм, маючи кінцевою метою виявлення файлів шкідливої програми.

Приховані прояви включають:

- наявність в пам'яті підозрілих процесів;
- наявність на комп'ютері підозрілих файлів;
- наявність підозрілих ключів в системному реєстрі *MS Windows*;
- підозріла мережева активність.

Ключовою ознакою у всіх випадках є атрибут "підозрілий". Тобто користувачеві невідоме призначення даного процесу, файла або ключа, і більш того, інформації про підозрілий об'єкт немає ні в документації до операційної системи, ні у відкритих джерелах мережі *Інтернет*.

Але перш ніж судити про підозрілість файлів і процесів, потрібно спочатку їх виділити із загального числа.

Підозрілі процеси – це фактично запущені виконувані файли. Частина процесів відноситься до операційної системи, частина до запущених програм. Знайти інформацію про невідомий процес можна в мережі *Інтернет*.

Автозапуск – особливою ознакою більшості черв'яків і багатьох троянських програм є зміна параметрів системи так, щоб файл шкідливої програми виконувався автоматично при кожному запуску комп'ютера. Тому наявність незнайомих файлів в списку файлів автозапуску також є приводом для пильного вивчення цих файлів. Залежно від настройок *MS Windows* і встановлених програм ключі автозапуску можуть містити різні рядки для запуску різних програм. Тому всі на перший погляд підозрілі файли потрібно перевіряти ще раз – вони можуть виявитися цілком звичайними програмами. У жодному випадку не слід змінювати налаштування системного реєстру навмання – це може призвести до повної неработоздатності комп'ютера і необхідності встановлювати заново операційну систему. Вносити зміни до реєстру можна тільки будучи абсолютно упевненим в своїх діях і повністю усвідомлюючи характер і наслідки модифікацій.

Мережева активність – шкідливі програми можуть виявлятися у вигляді мережевої активності. Черв'яки використовують мережу для розповсюдження, троянські програми – для завантаження додаткових компонентів і відсилання інформації зловмисникові. Деякі типи троянських програм спеціально призначені для забезпечення віддаленого

управління зараженим комп'ютером. Для забезпечення доступу до комп'ютера по мережі з боку зловмисника вони відкривають певний порт. Як правило, схожі за призначенням програми використовують одні і ті ж порти для прийому з'єднань. Аналогічно поштовому серверу, шкідливі програми для прийому команд або даних від зловмисника використовують певний порт, постійно чекаючи сигналів на цей порт. У таких випадках прийнято говорити, що програма слухає порт.

Отже, за наслідками аналізу процесів, параметрів автозавантаження і з'єднань отриманий список підозрілих з погляду користувача процесів (імен файлів). Для недосвідченого користувача невідомих, а значить підозрілих імен файлів може опинитися дуже багато, тому має сенс виділити найбільш підозрілі з них – ті, які були виявлені в двох і більш джерелах. Наприклад, файли, які присутні в списку процесів і в списку автозавантаження. Ще більш підозрілими є процеси, виявлені в автозавантаженні та такі, що прослуховують порти.

Для з'ясування природи підозрілих процесів найпростіше використовувати *Інтернет*. У глобальній мережі є сайти, що збирають інформацію про різні процеси. Даних по всіх процесах шкідливих програм на таких сайтах може і не бути, але в усякому разі на них є інформація про велику кількість безпечних процесів і таким чином можна буде виключити із списку ті процеси, які відносяться до операційної системи або відомих нешкідливих програм.

Після того, як список підозрілих процесів максимально звужений і в ньому залишилися тільки ті, про які немає вичерпної і достовірної інформації, залишається останній крок, знайти ці файли на диску і відправити на дослідження.

7.3 Методи захисту від шкідливих програм

Відомо, що для захисту від шкідливих програм потрібно використовувати антивіруси. Але в той же час нерідко можна почути про випадки проникнення вірусів на захищені антивірусом комп'ютери. У кожному конкретному випадку причини, з яких антивірус не справився зі своїм завданням можуть бути різні, наприклад:

- антивірус був відключений користувачем;
- антивірусні бази були дуже старі;
- були встановлені слабкі налаштування захисту;
- вірус використовував технологію зараження, проти якої у антивірусу не було засобів захисту;
- вірус потрапив на комп'ютер раніше, ніж був встановлений антивірус, і зміг знешкодити антивірусний засіб;
- це був новий вірус, для якого ще не були випущені антивірусні бази.

Але в цілому можна зробити висновок, що просто наявність встановленого антивірусу може бути недостатньо для повноцінного

захисту, і що потрібно використовувати додаткові методи.

Якщо поглянути, на наведені для прикладу причини пропуску вірусу антивірусом, можна побачити, що перші три причини пов'язані з неправильним використанням антивірусу, другі три – з недоліками самого антивірусу і роботою виробника антивірусу. Відповідно і методи захисту поділяються на два типи – організаційні і технічні.

7.3.1 Організаційні методи

Організаційні методи направлені насамперед на користувача комп'ютера. Їх мета полягає в тому, щоб змінити поведінку користувача, адже не секрет, що часто шкідливі програми потрапляють на комп'ютер через необдумані дії користувача. Простий приклад організаційного методу – розробка правил роботи за комп'ютером, яких повинні дотримувати всі користувачі.

На домашньому комп'ютері користувач сам встановлює собі правила, яким він вважає потрібним слідувати. У міру накопичення знань про роботу комп'ютера і про шкідливі програми, він може свідомо змінювати налаштування захисту або ухвалювати рішення про небезпеку тих або інших файлів і програм.

У великій організації все складніше. Коли колектив об'єднує велика кількість співробітників, що виконують різні функції, складно чекати від всіх розумної поведінки з погляду безпеки. Тому в кожній організації правила роботи з комп'ютером повинні бути загальними для всіх співробітників і затверджені офіційно. Зазвичай, документ, що містить ці правила називається інструкцією користувача. При цьому інструкція користувача в більшості випадків містить тільки правила, дії, що обмежують його. Правила використання програм в інструкцію можуть входити тільки в найбільш обмеженому вигляді.

Але якщо не користувачі, то хтось інший все-таки повинен відповідати за налаштування засобів захисту і за управління ними. Звичайно це спеціально призначений співробітник або група співробітників, які зосереджені на виконанні одного завдання – забезпечення безпечної роботи мережі.

Політика безпеки повинна давати відповіді на такі питання:

- які комп'ютери повинні бути захищені антивірусами та іншими програмами;
- які об'єкти повинні перевірятися антивірусом – чи потрібно перевіряти заархівовані файли, мережеві диски, вхідні і вихідні поштові повідомлення тощо;
- які дії повинен виконувати антивірус при виявленні зараженого об'єкта – іноді користувачі не можуть правильно вирішити, що робити з інфікованим файлом, отже антивірус повинен виконувати дії автоматично, не питаючи користувача.

7.3.2 Технічні методи

Технічні методи, навпаки, направлені на зміни в комп'ютерній системі. Більшість технічних методів полягають у використанні додаткових засобів захисту, які розширюють і доповнюють можливості антивірусних програм. Такими засобами захисту можуть бути:

- міжмережеві екрани – програми, що захищають від атак по мережі;
- засоби боротьби із спамом;
- виправлення, що знімають "дірки" в операційній системі, через які можуть проникати віруси.

Віруси нерідко проникають на комп'ютери через вразливості в операційній системі або встановлених програмах. Причому найчастіше шкідливими програмами використовуються вразливості операційної системи *MS Windows*, пакета додатків *MS Office*, браузера *Internet Explorer* і поштової програми *Outlook Express*.

Для того, щоб не дати вірусам можливості використовувати вразливість, операційну систему і програмне забезпечення потрібно оновлювати.

Хоча більшість шкідливих програм використовують вразливості в продуктах *Microsoft*, існує немало і таких, які експлуатують дірки в програмах інших виробників. Особливо це стосується широко поширених програм обміну повідомленнями, браузерів і поштових клієнтів. Тому мало просто встановити браузер, відмінний від *Internet Explorer*, його теж потрібно періодично оновлювати. Найчастіше, в подібних програмах є вбудовані засоби оновлення, але незайвим буде стежити за новинами на веб-сайтах, присвячених питанням безпеки. Інформація про всі критичні вразливості і виправленнях до них обов'язково потрапляє в новини.

Для того, щоб віддалено скористатися вразливістю в програмному забезпеченні або операційній системі, потрібно встановити з'єднання і передати спеціально сформований пакет даних. Отже, можна захиститися від таких спроб проникнення і зараження, шляхом заборони певних з'єднань. Задачу контролю з'єднань успішно вирішують *міжмережеві екрани*.

Міжмережевий екран – це програма, яка стежить за мережевими з'єднаннями і ухвалює рішення про дозвіл або заборону нових з'єднань на підставі заданого набору правил.

Правило міжмережевого екрана задається декількома атрибутами.

- **Додаток** – визначає програму, до якої відноситься правило, так що одні і ті ж дії можуть бути дозволені одним програмам і заборонені іншим.
- **Протокол** – визначає протокол, який використовується для передачі даних. Зазвичай можна вибрати між двома протоколами *TCP* і *UDP*.
- **Адреса** – визначає, для з'єднань з яких адрес або на які адреси діятиме правило.
- **Порт** – задає номери портів, на які розповсюджується правило.

• **Напрям** – дозволяє контролювати окремо вхідні та вихідні з'єднання.

• **Дія** – визначає реакцію на виявлення з'єднання, відповідно до перерахованих вище параметрів. Реакція може бути такою: дозволити, заборонити або запитати у користувача.

Не обов'язково давати конкретні значення всім атрибутам правила. Можна створити правило, яке заборонятиме вхідні з'єднання на порт 111 для всіх програмних додатків, або дозволяти будь-які вихідні з'єднання для програми *Internet Explorer*.

Для боротьби з вірусами міжмережеві екрани можуть застосовуватися двояко.

По-перше, міжмережевий екран можна успішно використовувати для захисту від шкідливих програм, які розповсюджуються безпосередньо по мережі, використовуючи вразливості в операційних системах. Міжмережеві екрани можна використовувати і для захисту від атак невідомих вірусів. У випадку домашнього комп'ютера, що використовує мережу тільки для доступу в *Інтернет*, можна заборонити взагалі всі вхідні з'єднання, і тим самим захистити себе від будь-яких атак ззовні.

Другий аспект застосування міжмережевих екранів для захисту від шкідливих програм полягає в контролі вихідних з'єднань. Багато троянських програм та черв'яків після виконання шкідливої функції прагнути подати сигнал авторові вірусу. Для того, щоб перешкодити цьому, можна побудувати міжмережевий екран таким чином, щоб він блокував всі невідомі з'єднання: дозволити тільки з'єднання від довірених програм, таких як використовуваний браузер, поштовий клієнт, а решту з'єднань заборонити. У такому разі, шкідлива програма, навіть потрапивши на комп'ютер непоміченою, не зможе відправляти дані.

Деякі шкідливі програми не намагаються активно пересилати дані, а пасивно чекають з'єднання на якомусь з портів. Якщо вхідні з'єднання дозволені, то автор шкідливої програми зможе через деякий час звернутися на цей порт і забрати потрібну йому інформацію або ж передати шкідливій програмі нові команди. Щоб цього не відбулося, міжмережевий екран повинен бути налаштований на заборону вхідних з'єднань або на всі порти взагалі, або на всі, окрім фіксованого переліку портів, які використовують відомі програми або операційна система.

Останнім часом широко поширені універсальні захисні програми, які об'єднують можливості міжмережевого екрана та антивірусу.

7.4 Тестовий вірус

Завершальний етап установлення будь-якої програми – це перевірка коректності виконання основних її функцій. Для антивірусних додатків основу функціонала становить здатність знаходити й знешкоджувати шкідливі програми.

Природно, постає питання як перевірити чи дійсно програма може це робити – адже відомо, що нові віруси з'являються щодня, причому десятками, а іноді й сотнями. Не кожному користувачеві під силу регулярно відслідковувати хоча б їхню частину. Цим займаються антивірусні компанії. Їхні філії, розкидані по всьому світі, безупинно стежать за вірусною активністю в *Інтернет*, перехоплюють й аналізують всі підозрілі файли. На основі отриманих даних формуються вірусні сигнатури, які користувач одержує під час відновлення своїх антивірусних баз. Таким чином, перевірити надійність антивірусного захисту від всіх уже існуючих вірусів і тих, які тільки завтра або через рік будуть створені, нереально. До того ж, використати дійсні віруси тільки для попереднього тестування програми не можна. Не можна виключати ймовірність, що програма установлення десь дала збій й отже захист не встановлений. Тоді під час перевірки може відбутися зараження вірусом, на якому виробляється тестування, що неприпустимо.

Але незважаючи на всі ці проблеми, метод діагностики антивірусних програм все-таки існує. Для цього використається спеціальний файл, "*The Anti-Virus or Anti-Malware test file*", створений Європейським інститутом комп'ютерних антивірусних досліджень (*European Institute for Computer Antivirus Research – EICAR*).

Тестовий вірус, розроблений Європейським інститутом комп'ютерних антивірусних досліджень, називається – за абревіатурою повної назви інституту *EICAR*.

EICAR являє собою невеликий 68-байтовий файл. Його розширення можна варіювати залежно від сценарію тестування. Якщо додати *.com*, то запуск файла, що вийшов, *eicar.com* на незахищеному комп'ютері викличе тільки показ повідомлення "*EICAR-STANDARD-ANTIVIRUS-TEST-FILE!*". Інших, властивих вірусам проявів, він не несе. Однак якщо на комп'ютері стоїть й справно працює антивірус, *EICAR* буде заблокований. Це відбувається тому, що всі провідні виробники антивірусних програм домовилися між собою – вважати *EICAR* вірусом і застосовувати до нього всі правила й дії, застосовувані до дійсних шкідливих програм.

Для більш докладного тестування можна застосовувати інші розширення. Наприклад, якщо вказати *.txt*, можна перевірити чи перевіряються текстові файли. Для перевірки чи будуть виявлятися віруси в архівах, *EICAR* можна заархівувати.

Якщо відкрити *EICAR* у будь-якому текстовому редакторі, наприклад *Блокнот*, то виявиться, що він складається з 68 символів:

```
X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-
  TEST-FILE!$H+H*
```

Отже, тестовий вірус у будь-який момент можна створити самостійно. Для цього потрібно тільки відкрити будь-який текстовий редактор, набрати в ньому цей рядок і зберегти файл, що вийшов, у форматі текстового файла (звичайний текст).

EICAR також завжди можна завантажити із сайта Європейського інституту комп'ютерних антивірусних досліджень http://www.eicar.org .

Суть *EICAR* така, що він виявляється невиліковним. Це відбувається тому, що антивірус ідентифікує *EICAR* як вірус за наявністю в ньому згаданих 68 символів. Якщо їх видалити – то від файла нічого не залишиться. Отже, за допомогою *EICAR* можна тестувати тільки основну функцію антивірусу – виявлення.

Тому для тестування своїх продуктів компанія *Лабораторія Касперского* пропонує використати модифікований тестовий вірус, варіації якого показані в таблиці 7.1.

Таблиця 7.1 – Модифікований тестовий вірус *EICAR*

Файл	Опис
CURE-EICAR	Виявивши такий файл, антивірус повинен його "вилікувати", скоротивши його розмір до 4 байт (символи "*CURE*")
DELE-EICAR	Цей файл Антивірус Касперского визначає як невиліковний вірус або троянську програму й видаляє. Отже, за результатами перевірки *DELE-EICAR* повинен бути виявлений тільки в резервному сховищі
CORR-EICAR	Призначений для діагностики роботи Антивірусу Касперского у випадку виявлення файла з ушкодженою структурою, внаслідок чого перевірити його на наявність вірусів неможливо. Такий файл визнається умовно чистим
ERRO-EICAR	При скануванні такого файла Антивірус Касперского повинен поводитися так, начебто відбулася помилка при аналізі його вмісту (наприклад, через порушення цілісності при перевірці багатотомного архіву або при обриві зв'язку під час перевірки по мережі). *ERRO-EICAR* також визнається умовно чистим
SUSP-EICAR	Цей файл коректно працюючий Антивірус Касперского визнає підозрілим, а саме: зараженим невідомим вірусом. Отже, він повинен бути поміщений на карантин або вилучений (залежно від настроювань, за замовчуванням дія при виявленні підозрілого об'єкта запитується в користувача)
WARN-EICAR	*WARN-EICAR* також визнається підозрілим, але не невідомим вірусом, а модифікацією відомого. Це також приводить до пропозиції помістити його на карантин або видалити (залежно від настроювань)

Створюються ці файли за таким принципом. На початок 68-символьного рядка поміщається п'ять символів, залежно від модифікації – *CURE*, *DELE*, *CORR*, *ERRO*, *SUSP* або *WARN* та дефіс.

8 ПРАКТИЧНІ РОБОТИ

8.1 Перевірка сертифікатів комп'ютерної безпеки

1. Визначіть, які операційні системи використовуються у комп'ютерному класі на робочому комп'ютері.

2. Скопіюйте класи безпеки відповідно до "Оранжевої книги" з веб-сайту за адресою *http://www.radi-um.ncsc.mil/tpep/library/rainbow/* або з *Дисципліни/4 курс/Захист комп'ютерних мереж.*

3. Почніть з перевірки функціональних вимог розділу III "Оранжевої книги". На цьому етапі ігноруйте вимоги гарантованості і документування.

4. Визначіть, чи відповідає дана система вимогам розділу С. Якщо це так, то переходьте до розділів В і А.

5. Після визначення функціонального рівня системи перевірте вимоги гарантованості і документування для цього ж рівня. Чи виконуються ці вимоги?

6. Проаналізуйте інформацію, що знаходиться на вашому комп'ютері. Виявіть найважливішу. Визначіть місце зберігання цієї інформації

7. Визначіть типи атак, найбільш руйнівних для вас. Продумайте вірогідність атаки доступу, атаки модифікації, атаки на відмову в обслуговуванні.

8. Продумайте способи виявлення таких атак.

9. Виберіть той тип атаки, яка, на вашу думку, є найбільш руйнівною, і розробіть стратегію атаки.

Рекомендації

Рівень С2 ґрунтується на вимозі безпеки повторного використання об'єкта, і більшість комерційних операційних систем відповідають вимогам функціональності цього рівня. Ці системи не мають функціональності, відповідної рівню С1.

Вимоги гарантованості і документування навіть для рівня С1 навряд чи можна зустріти в стандартній документації до програмного забезпечення.

Для багатьох комерційних компаній найбільш секретними відомостями є картотека персоналу і інформація про заробіток. Не потрібно забувати про покупців – про їх номери кредитних карт і номери соціального забезпечення. Фінансові організації і організації охорони здоров'я також мають секретну інформацію, яка певним чином регулюється. Проглядаючи інформацію і продумувавши можливість атаки, поставте за мету зробити так, щоб вона не була розкрита. Можливо, що для вашого бізнесу важливо врахувати атаки модифікації, відмови в обслуговуванні і відмови від зобов'язань.

8.2 Аналіз основних загроз інформаційних потоків підприємства

1. Проведіть загальний аналіз структури підприємства. Проаналізуйте загальну кількість працівників, відділів та їх взаємодію. Розробіть структурну схему підприємства з указанням всіх відділів та працівників, які в них працюють, їх посади та функціональні обов'язки, пов'язані з використанням інформаційних технологій та комп'ютерних мереж.

2. Проведіть аналіз інформаційних потоків, які існують всередині підприємства. Окремо виділіть зовнішні та внутрішні потоки, а також інформаційні потоки, які використовуються для обміну даними між різними відділами підприємства. Проаналізуйте характер інформаційних потоків між різними відділами та працівниками, виділяючи інформаційні потоки з інформацією, що потребує захисту, визначте її статус (комерційна таємниця, службова інформації, фінансова інформація, лише для внутрішнього використання, загальнодоступна тощо). На схемі необхідно показати окремі потоки інформації, що потребують захисту.

3. Як результат проведеного аналізу наведіть таблицю розмежування доступу для різних категорій працівників або відділів.

4. Розробіть політику безпеки верхнього рівня підприємства в галузі інформаційних технологій, в тому числі стосовно підключення до глобальної мережі Інтернет.

5. Зробіть висновки стосовно необхідності розробки комп'ютерної мережі та захисту окремих інформаційних потоків.

8.3 Захист *IP*

1. Під час обговорення протоколу *AH,* який використовується в стеці протоколів *IPSec,* було сказано, що не всі поля в заголовку *IP* беруть участь в обчисленні значення *MAC*:

а) для кожного поля в заголовку *IPv4* укажіть, чи буде воно незмінним, змінюваним, але прогнозованим, або просто змінюваним (значення останнього перед обчисленням значення *ICV* (*Integrity Check Value*) повинне обнулятися);

б) зробіть те ж саме для заголовка *IPv6*;

в) виконайте те ж для заголовків розширень *IPv6*.

У кожному випадку обґрунтуйте свій вибір для кожного з полів.

2. При використанні тунельного режиму створюється новий зовнішній заголовок *IP*. Для *IPv4* і *IPv6* зв'яжіть кожне поле зовнішнього заголовка *IP* і кожен заголовок розширення в зовнішньому пакеті з відповідними полями або заголовками розширень внутрішнього пакета *IP*. Іншими словами, укажіть, які зовнішні значення задаються на основі внутрішніх значень, а які створюються незалежно.

3. При зв'язку між двома кінцевими системами бажані автентифікація і шифрування. Зобразіть, подібно тому, як це зроблено на

рис. 3.6 та 3.9, структури відповідно до поставлених завдань:

а) транспортну суміжність із шифруванням, застосовуваним до автентифікації;

б) захищений транспортний зв'язок, вкладений у захищений тунельний зв'язок, із шифруванням, застосовуваним до автентифікації;

в) захищений транспортний зв'язок, вкладений у захищений тунельний зв'язок, з автентифікацією, застосовуваною до шифрування.

4. У документі, що описує архітектуру *IPSec*, стверджується, що в комбінації двох захищених зв'язків у транспортному режимі з застосуванням протоколів *AH* і *ESP* для одного наскрізного потоку прийнятним виявляється тільки одне упорядкування протоколів захисту – використання протоколу *ESP* до застосування протоколу *AH*. Чому рекомендується саме такий порядок, а не автентифікація до шифрування?

5. Який з типів обміну *ISAKMP* відповідає енергійному обмінові ключами *Oakley*?

6. Для повідомлень енергійного обміну ключами *Oakley* укажіть, до якого з типів корисного вантажу *ISAKMP* відноситься кожний з параметрів відповідного повідомлення.

8.4 Протокол захищених транзакцій

1. Чому в протоколах *SSL* і *TLS* передбачене використання окремого протоколу зміни параметрів шифрування (*Change Cipher Spec Protocol*), а не просто повідомлення *change_cipher_spec* у рамках протоколу квітерування (*Handshake Protocol*)?

2. Розгляньте такі загрози безпеки *Web* і поясніть, як кожна з них усувається тими або іншими засобами *SSL*.

- Криптоаналіз на основі перебору усіх варіантів. Перебір усіх можливих варіантів ключів алгоритму традиційного шифрування.
- Аналіз з використанням словника відомого відкритого тексту. Багато повідомлень містять відкритий текст певного виду, наприклад, команди *GET HTTP*. Зловмисник створює словник, що включає всі можливі варіанти шифрування відомого відкритого тексту. Перехопивши шифроване повідомлення, супротивник бере фрагмент шифрованого повідомлення і шукає в ньому відповідний шифрований текст зі словника. Відповідний текст словника і текст фрагмента повинні бути зашифровані одним і тим самим секретним ключем. Якщо виявлено кілька збігів, починаються спроби дешифрувати текст із кожним з відповідних ключів, щоб з'ясувати, який з них правильний. Цей метод аналізу особливо ефективний при малій довжині ключа (до 40 біт).
- Відтворення повідомлень. Зловмисник відтворює раніше перехоплені ним повідомлення квітерування *SSL*.
- Впровадження посередника. Зловмисник впроваджується в лінію обміну ключами між клієнтом і сервером як посередник,

представляючись клієнтові сервером, а серверові – клієнтом.

- Крадіжка пароля. Перехоплення пароля зловмисником у потоці даних додатків *HTTP* або іншого додатка.
- Фальсифікація *IP*-адреси. Використання фальсифікованих *IP*-адрес при спробі змусити головний вузол прийняти помилкові дані.
- Захоплення *IP*-зв'язку. Відсторонення однієї з легальних сторін поточного автентифікованого з'єднання шляхом її відключення з одночасним заміщенням її зловмисником.
- Синхронна атака. Зловмисник відправляє повідомлення *SYN* (символ синхронізації) протоколу *TCP* із запитом на установлення з'єднання, але не відповідає на завершальне повідомлення, зберігаючи тим самим з'єднання відкритим. Модуль *TCP*, що атакується, звичайно очікує завершення процесу відкриття "напіввідчиненого" з'єднання протягом деякого часу, так що багатократно повторювані повідомлення *SYN* можуть цілком паралізувати роботу модуля *TCP*.

Грунтуючись на матеріалах лабораторних робіт, чи можете ви стверджувати, що одержувач має можливість відновити вихідний порядок записів *SSL*, що надійшли, якщо вони прийшли з порушенням цього порядку? Якщо так, то як це зробити? Якщо ні, то чому?

8.5 Протокол мережевого керування

1. Протокол *SNMPv1* визначає тип даних, називаний *датчиком,* і містить таке пояснення семантики цього типу.

«Цей тип являє собою від'ємне ціле число, що може зростати або зменшуватись, але яке замикається *(latch)* при максимальному значенні. Даний стандарт задає максимальне значення для датчиків, рівне 2^{31}-1 (або 4294967295 у десятковому поданні).»

На жаль, тут слово *latch* (замикати) не одержало точного означення, що породило дві його різні інтерпретації. Стандарт *SNMPv2* усуває неоднозначність за допомогою такого означення.

«Значення датчика виявляється рівним максимальному значенню, якщо модельована ним інформація більша або дорівнює цьому максимальному значенню; якщо ж згодом модельована датчиком інформація зменшується нижче максимального значення, значення датчика також зменшується.»

а) яка альтернативна інтерпретація?

б) розглянути всі "за" і "проти" цих двох інтерпретацій.

2. У *SNMPv1* будь-який об'єкт *MIB* має категорію доступу *MIB*, що може приймати одне з таких значень: тільки для читання, запис-читання-запис, тільки запис і недоступний. Читання виконується при операціях *get* і *trap*, а запис – при операціях *set.* Для читання-запису об'єкт може бути доступний при виконанні операцій *get* та *trap*, але це залежить від реалізації. Категорія доступу *MIB* описує максимум можливостей доступу для об'єкта, але в рамках можливостей об'єднань *SNMPv1* режим доступу може додатково обмежувати доступ за допомогою профілю даного

об'єднання. У таблиці заповніть порожні комірки, указавши відповідний вид дозволеного доступу.

Категорія доступу MIB	Режим доступу SNMP	
	ТІЛЬКИ	ЗАПИС-ЧИТАННЯ-ЗАПИС
Тільки читання		
запис-читання-запис		
тільки запис		
недоступний		

3. У *SNMPv1* є такі особливості:

а) для неавторитетного процесора у заголовку вихідного повідомлення значення *msgAuthoritativeEngineBoots* і *msgAuthoritativeEngineTime* встановлюються тільки у випадку, якщо повідомлення повинно автентифікуватися авторитетним одержувачем. Чому це обмеження необхідне?

б) однак для повідомлення *Response* авторитетного процесора значення *msgAuthoritativeEngineBoots* і *msgAuthoritativeEngineTime* у заголовку вихідного повідомлення встановлюються завжди. Чому?

4. Синхронізація часу (відновлення показань локального часу на основі значень, що надходять) виконується до верифікації вікна часу (перевірки того, що повідомлення, яке надійшло, своєчасне). Це означає, що неавторитетний процесор може обновити своє подання часу авторитетного процесора, якщо отримане повідомлення автентифіковано, навіть якщо повідомлення не попадає в рамки припустимого вікна часу. Корисно буде розглянути деякі наслідки цього. Визначимо такі скорочення.

`TPABHIB=msgAuthoritativeEngineBoots;MAET=msgAuthoritativeEngineTime;`

`SEB` = локальне подання *snmpEngineBoots* для віддаленого авторитетного процесора;

`SET` = локальне подання *snmpEngineTime* для віддаленого авторитетного процесора;

`LRET = latestReceivedEngineTime.`

Тепер припустимо, що неавторитетний процесор одержує повідомлення вигляду

`(TPABHIB = SEB) AND [LRET < MAET < (SET - 150)].`

Тоді умови оновлення годинника виконуються, тому відбувається таке:

`SET := MAET; LRET := MAET.`

Тепер, коли необхідно перевірити вікно часу, ми маємо

`(TPABHIB = SEB) AND (MAET = SET),`

так що ми можемо заявити, що повідомлення є своєчасним. Припустимо, однак, що ми спочатку повинні виконати перевірку вікна часу. Яким тоді ми повинні визнати повідомлення – своєчасним чи ні?

5. У першій опублікованій версії специфікацій *USM* під час

обговорення процедури синхронізації часу і верифікації вікна часу робиться таке зауваження. "Зверніть увагу на те, що ця процедура не дозволяє виконати автоматичну синхронізацію часу, якщо неавторитетний процесор *SNMP* виявляється в ситуації, коли авторитетний процесор *SNMP* відстає від неавторитетного процесора більше ніж на 150 секунд." Це твердження не увійшло у виправлену версію після того, як робочій групі було зазначено, що це твердження не завжди правильне. Використовуючи приклад із задачі 4, доведіть, що це твердження не завжди правильне.

6. *SNMPv3* припускає що є деякі захищені засоби доставки локалізованих ключів автентифікованим системам (агентам). Така захищена доставка не є предметом розгляду протоколу *SNMPv3* – це може бути доставка вручну або за допомогою деякого захищеного протоколу, відмінного від *SNMPv3*. Після доставки агентові вихідного ключа (або пари ключів – автентифікації і таємності) *SNMPv3* надає механізм захищеного відновлення ключів. Для посилення захисту потрібно час від часу змінювати ключі. Користувач, зі своєї сторони, може ініціювати процес зміни ключів за допомогою запиту і надання нового пароля. Система мережевого керування *(Network Management System – NMS)* може ініціювати цей процес за допомогою запиту нового пароля. У будь-якому випадку ключ користувача в *NMS* змінюється. Після цього *NMS* може обчислити локалізовані ключі для кожного зі своїх агентів. Система *NMS* повинна потім зв'язатися захищеним чином з кожним агентом, щоб змусити агента змінити локалізований ключ. Ясно, що *NMS* не може просто переслати ключ у відкритому вигляді мережею. Очевидні такі два варіанти.

- Шифрувати новий ключ, використовуючи старий як ключ шифрування.
- Використовувати деяку однобічну функцію, щоб згенерувати деяке значення зі старого ключа. Зв'язати за допомогою операції *XOR* (виключне АБО) це значення з новим ключем і послати результат агентові. Агент може потім за допомогою операції *XOR* зв'язати значення, що надійшло, з результатом застосування тієї ж однобічної функції до старого ключа, щоб одержати новий ключ.

У *SNMPv3* використовується варіант іншого методу. Які переваги цього методу в порівнянні з першим?

7. Підхід *SNMPv3* припускає використання об'єкта *KeyChange* (зміна ключа) у структурі *MIB* системи призначення. Віддалений принципал або *NMS* установлює цей об'єкт, а потім він автоматично використовується агентом для відновлення відповідного ключа. Алгоритм складається з двох частин, одна з яких існує в процесорі, що надсилає запит, а інша – у віддаленому процесорі агента. Процес починається, коли сторона-відправник виявляє бажання змінити існуючий ключ *keyOld* на нове значення *keyNew*. Сторона-відправник виконує такі дії.

7.1. Генерує випадкове значення *random* за допомогою генератора псевдовипадкових або істинно випадкових чисел.

7.2. Обчислюється `digest = Hash (keyOld || random)`,

де *Hash* позначає або *MD5*, або *SHA-1*, у залежності від того, який потрібен ключ (16- або 20-байтовий), а символ || позначає конкатенацію.

7.3. Обчислюються

`delta = digest⊕keyNew, protocolKeyChange = (random || delta)` ,

де ⊕ позначає операцію „*XOR*".

7.4. Значення *protocolKeyChange* потім посилається агентові в команді *Set* для відновлення екземпляра об'єкта *KeyChange* у структурі *MIB* агента. Що повинен зробити агент зі значенням, яке надійшло, щоб змінити ключ?

8. Більш проста процедура (у порівнянні з тільки що описаною) могла б припускати зв'язування *keyOld* і *keyNew* за допомогою операції *XOR* і таку передачу отриманого в результаті значення. Одержувач потім може зв'язати операцією *XOR* вхідне значення і *keyOld*, щоб одержати *keyNew*. Оскільки зловмисник не знає значення *keyOld*, він не зможе обчислити значення *keyNew*. У чому полягають переваги використання випадкових чисел і захищеної однобічної функції хешування за схемою з задачі 7 у порівнянні з підходом, запропонованим тут?

8.6 Протоколи *SSL* та *TLS*

1. Опишіть структуру протоколу *SSL*. В чому особливість його використання? Наведіть приклади використання протоколу *SSL*.

2. Опишіть структуру протоколу *TLS*. В чому особливість його використання? Наведіть приклади використання протоколу *TLS*.

3. Дайте відповідь на запитання: Чому в протоколах *SSL* та *TLS* передбачене використання окремого протоколу зміни параметрів шифрування (*Change Cipher Spec Protocol*), а не просто повідомлення *change_cipher_spec* у рамках протоколу квітерування (*Handshake Protocol*)?

4. Розгляньте такі загрози безпеці *Web* і поясніть, як кожна з них усувається тими чи іншими засобами *SSL*:

- криптоаналіз на основі перебору усіх варіантів;
- аналіз з використанням словника відомого відкритого тексту;
- відтворення повідомлення;
- крадіжка пароля;
- фальсифікація *IP*-адреси;
- захоплення *IP*- зв'язку;
- синхронна атака.

8.7 Використання криптографічного захисту електронної пошти

Використовуючи теоретичні відомості наведені в главі 6 дайте

обґрунтовану відповідь на такі запитання:

1. Який криптографічний захист файлів і повідомлень (шифрування і підписка) використовується для захисту електронної пошти?

2. Як відбувається підписка файлів без їх зашифрування?

3. Як відбувається розшифрування файлів і перевірка їх цифрових підписів?

4. В чому полягає гарантоване видалення файлів?

5. Який криптографічний захист повідомлень електронної пошти (шифрування і підписка)?

6. Яким чином відбувається розшифровка і перевірка підпису отриманих повідомлень?

9 ЛАБОРАТОРНІ РОБОТИ

9.1 Лабораторна робота №1 – Виявлення атак

Мета роботи. Навчитися визначати *IP*-адреси веб-сервера, поштового сервера та іншої інформації. Виявити можливі шляхи атаки інформації або комп'ютерної системи.

Хід роботи

Виявлення атак – справа нелегка. Для цього можна використовувати електронні засоби, але не можна нехтувати проблемами фізичної безпеки і персоналом організації. Чи звернув адміністратор компанії увагу на те, що в обчислювальному залі був сторонній? Чи відмітив адміністратор зміну файла?

Нарешті при виробленні стратегії не обмежуйтеся комп'ютерами і мережами. Подумайте про те, як зловмисник може використовувати фізичні засоби для отримання інформації або для її знищення.

1. Визначите *IP*-адресу веб-сервера нашого інституту. Введіть в командному рядку

```
nslookup <имя_веб-сервера>.
```

Повернуте значення і буде *IP*-адресою веб-сервера.

2. Визначте *IP*-адресу поштового сервера.

У командному рядку введіть

```
nslookup.
```

Після запуску програми введіть

```
set type=mx
```

і натисніть **[Enter]**.

Потім введіть

```
<ім'я вашого домену>
```

і натисніть **[Enter]**.

Програма поверне список основних і додаткових поштових серверів.

3. Перейдіть за допомогою браузера за адресою http://www.arin.net/ і в рядку пошуку введіть адреси.

Повернеться інформація про того, хто володіє блоком адрес. Тепер ви маєте непогане уявлення про блоки адрес, призначених вашій організації, а за наявності хостінгу – адреси головного веб-сайта.

4. Введіть в рядку пошуку ім'я організації і отримаєте список всіх призначених *IP*-адрес.

5. Перейдіть за допомогою браузера за адресою http://www.network-solutions.com/ і введіть в рядку пошуку ім'я свого домену. Отримайте інформацію про розміщення своєї організації за допомогою видачі списку з'єднань. Тепер ви знаєте основні *DNS*-сервери, які обслуговують ваш домен.

6. Відправте *ping*-пакети та проскануйте адресний простір. Дізнаєтеся про те, які хости зараз знаходяться в режимі онлайн. Пам'ятаєте про те, що за наявності міжмережевого екрана сканування портів займе деякий час.

Зміст звіту

Користуючись теоретичними відомостями наведеними в розділі 1 та методичними вказівками до експериментальної частини виконати поставлене завдання.

Скласти звіт з роботи.

Контрольні запитання

1. Прикладом якого вразливого місця є в *NFS* установлення дозволів кореневій директорії *rw* для всіх користувачів?
2. Коли використовуються нетехнічні засоби для діставання доступу в систему?
3. Яка частина пам'яті є об'єктом атаки на переповнювання буфера?
4. Який тип змінних використовується при виконанні атаки на переповнювання буфера?
5. Яка помилка програмування дозволяє виконати атаку імітації *IP*-адреси?
6. Який пакет не відправляється при виконанні синхронної атаки?
7. Чи існує спосіб захисту від грамотно розробленої *DoS*-атаки?
8. Що шукають зловмисники, які використовують ненаправлені методи атак?
9. Як зловмисник використовує систему після злому за допомогою ненаправленої атаки?
10. Який сайт використовується для збирання інформації про *IP*-адреси?
11. Яка частина попереднього дослідження є найбільш небезпечною для зловмисника при підготовці направленої атаки?
12. З якою метою запускається атака *DoS* під час виконання атаки імітації *IP*-адреси?
13. Чим є програма "Троянський кінь" для користувача?

9.2 Лабораторна робота №2 – Захист на рівні комунікаційних протоколів

Мета роботи. Зробити огляд можливостей захисту на рівні комунікаційних протоколів, вивчивши їх архітектуру, принципи адресації, маршрутизації.

Підготовка до роботи

Вміст *ARP*-кеша можна побачити за допомогою використання команди *arp*. Опція «*–a*» показує всі записи, що містяться в кеші. Для виконання цієї команди ввійдіть у меню «*Пуск*» (середовище *Windows* сімейства *NT*), клацніть пункт «*Виконати*», введіть *cmd.exe* для запуску командного інтерпретатора і в ньому введіть команду «*arp –a*» для того, щоб показати всі активні записи кеша *ARP* (рис. 9.1).

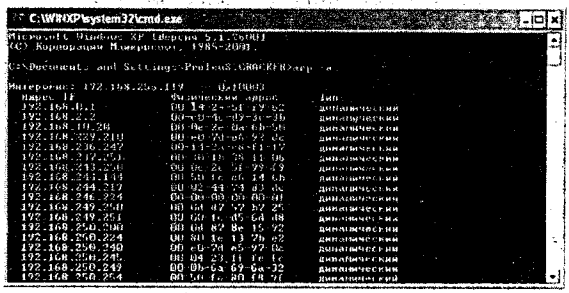


Рисунок 9.1 – Вікно перегляду активних записів arp-кешу

Нові статичні записи можна додавати командою «*arp -s new_ip new_MAC*», де *new_ip* – *IP*-адреса нової машини в мережі, а *new_MAC* – *MAC*-адреса її мережевої карти. Цей спосіб дозволяє перешкодити атакам *ARP-spoofing*, унеможливлюючи підміну записів у системній *ARP*-таблиці.

Хід роботи

1. Обчисліть максимально можливу кількість комп'ютерів у мережах класу *A*, *B* й *C*.
2. Перегляньте облікові записи користувачів за допомогою утиліти *net*.
3. Перегляньте всі відкриті ресурси за допомогою команди *net share*.
4. Створіть новий мережевий ресурс.
5. Перегляньте список комп'ютерів у поточному домені за допомогою команди *net view*.
6. Відішліть повідомлення на інший мережевий комп'ютер за допомогою команди *net send*.
7. Перегляньте всі динамічні записи системної *ARP*-таблиці.

8. Додайте в неї 1 статичний запис із випадковими *IP* та *MAC*-адресами.

Зміст звіту

Користуючись теоретичними відомостями наведеними в розділі 2 та методичними вказівками до експериментальної частини виконати поставлене завдання.

Скласти звіт з роботи.

Контрольні запитання

1. Дайте означення *ARP*-протокола.
2. Які атаки можна реалізувати, використовуючи недоліки *ARP*-протоколу?
3. Що таке *ARP*-таблиця? Як її переглянути?
4. Які атаки можна реалізовувати на рівні комунікаційних протоколів?
5. Яким чином можна захиститися від атак на рівні комунікаційних протоколів?
6. Чому виникла необхідність у введені протоколу *IPv6*?
7. Для чого використовується служба *DNS*? Які атаки можна реалізувати на цю службу?

9.3 Лабораторна робота №3 – Захист *Web*

Мета роботи. Зробити огляд технологій атак рівня *Web* та можливостей захисту від них, вивчивши принципи їх реалізації. Формування умінь використання *Cookies*.

Підготовка до роботи

Опираючись на матеріал теоретичних відомостей, наведений в розділі 4, вивчити методи захисту *Web* від описаних видів атак.

Короткі теоретичні відомості

Cookie є вирішенням однієї із спадкових проблем *HTTP* специфікації. Ця проблема полягає в непостійності з'єднання між клієнтом і сервером, як при *FTP* або *Telnet* сесії, тобто для кожного документа (або файла) при передачі по *HTTP* протоколу посилається окремий запит. Включення *cookie* в *HTTP* протокол дало часткове вирішення цієї проблеми.

Cookie це невелика порція інформації, яку сервер передає клієнтові. Клієнт (браузер) зберігатиме цю інформацію, і передавати її серверу з кожним запитом як частине *http*-заголовка. Деякі *cookie* зберігаються тільки протягом однієї сесії, вони видаляються після закриття браузера. Інші, встановлені на деякий період часу, записуються у файл. Ці файли ви можете бачити у себе на комп'ютері в каталозі *\Documents and Settings\Имя_пользователя\Cookies*.

Самі по собі *cookies* не можуть робити нічого, це тільки деяка інформація. Проте, сервер може реагувати на інформацію, що міститься в *cookies*. Наприклад, у разі авторизованого доступу через *WWW*, в *cookies* можна зберігається *login* і *password* протягом сесії, що дозволяє не вводити їх при запиті кожного документа. Інший приклад: *cookies* можуть використовуватися для побудови сторінок, що персоналізуються. Найчастіше зустрічається таке – на деякому сервері просять ввести своє ім'я, і кожного разу, при заходженні на першу сторінку цього сервера, пишуть щось типу "Здрастуйте, Вася Пупкін!". При використанні *cookies* також часто будують функцію оформлення замовлення в онлайнових магазинах.

Cookie є частиною *HTTP*-заголовка. Повний опис поля *Set-Cookie HTTP* заголовка:

```
Set-Cookie:NAME=value;EXPIRES=date;DOMAIN=domain_name; PATH=path; SECURE
```

Мінімальний опис поля *Set-Cookie HTTP*-заголовка:

```
Set-Cookie: NAME=VALUE;
```

`NAME=VALUE` – `NAME` – змінна (ім'я cookie), `VALUE` – значення.

`expires=DATE` – час зберігання cookie (формат запису "`expires=Monday, DD-Mon-YYYY HH:MM:SS GMT`").

Приклад:

```
expires=Friday,31-Dec-99 23:59:59 GMT;
```

Якщо цей атрибут не вказаний, то *cookie* зберігається протягом одного сеансу, до закриття браузера.

`domain=DOMAIN_NAME` – домен, для якого значення cookie дійсне.

Наприклад, "*domain=ipm.kstu.ru*".

Якщо цей атрибут опущений, то за замовчуванням використовується доменне ім'я сервера, на якому було задано значення *cookie*.

`path=PATH` – цей атрибут встановлює шлях до документів, для яких дійсне значення *cookie*. Наприклад, вказівка *"path=/students/pupkin/"* приведе до того, що значення *cookie* буде дійсне для всіх документів Васі Пупкіна в директорії */students/pupkin/*. Для того, щоб cookie відсилалися при запиті будь-якого документа сервера, необхідно вказати кореневий каталог сервера ("*path=/*").

Якщо цей атрибут не вказаний, то значення *cookie* поширюється тільки на документи в тій же директорії, що і документ, в якому було встановлено значення *cookie*.

secure – якщо стоїть цей маркер, то інформація cookie пересилається тільки через *HTTPS* (*HTTP* з використанням *SSL – Secure Socket Level*), в шифрованому вигляді. Якщо цей маркер не вказаний, то інформація пересилається звичайним способом.

Встановлення *cookie* за допомогою *HTML*.

Простий спосіб виставити *cookie* – використовувати відповідний META-тег в заголовку *<HEAD> </HEAD>* будь-якого *HTML* документа. Це виглядає таким чином:

```
<META HTTP-EQUIV="Set-Cookie" CONTENT="NAME=value;
 EXPIRES=date; DOMAIN=domain_name; PATH=path; SECURE">
```

За допомогою *HTML* можна встановити декілька *cookie* разом:

```
<META HTTP-EQUIV="Set-Cookie" CONTENT="NAME1=value1; EXPIRES=date1;
DOMAIN=domain_name1; PATH=path1; SECURE"> <META HTTP-EQUIV="Set-Cookie"
CONTENT="NAME2=value2; EXPIRES=date2; DOMAIN=domain_name2; PATH=path2;
SECURE">
```

Коли запитується документ з *http*-сервера, браузер перевіряє свої *cookie* на предмет відповідності домену сервера (*DOMAIN*), шляху до документа (*PATH*), якщо час зберігання закінчився, то він віддаляється. У випадку, якщо знайдені значення *cookie*, що задовольняють всі умови, браузер посилає їх серверу у вигляді пари ім'я/значення:

```
Cookie: NAME1=STRING1; NAME2=STRING2 ...
```

У цьому випадку *cookie* набуває нового значення при наявному вже в браузері *cookie* із збіжними параметрами *NAME*, *domain* та *path*, то старе значення замінюється новим. У решті випадків нові значення *cookie* додаються до старих.

Використання *expires* не гарантує збереження *cookie* протягом заданого періоду часу, оскільки клієнт (браузер) може видалити запис через нестачу виділеного місця або будь-які інші причини.

Клієнт (браузер) може мати обмеження для *cookies*, наприклад:

- всього може зберігатися до 300 значень *cookies*;
- кожен *cookie* не може перевищувати 4 Кбайт;
- з одного сервера або домену може зберігатися до 20 значень *cookie*.

Якщо обмеження 300 або 20 перевищується, то віддаляється перший за часом запис. При перевищенні ліміту об'єму в 4 Кбайт коректність значення *cookie* страждає – відрізується частина запису (з початку цього запису) рівна перевищенню об'єму.

За наявності *SSI* на сервері, директивою `<!--#echo var="..."-->` можна враховувати будь-які змінні оточення, у тому числі і раніше задані значення *cookie* (змінна *HTTP_COOKIE*).

Приклади:

```
<!--#echo var="HTTP_COOKIE"-->
```

Вставлення всіх отриманих *cookies* в документ.

```
<!--#if expr="$HTTP_COOKIE=/user\=old/" -->
```

Перевірка чи міститься в *cookies* "`user=old`".

```
<META HTTP-EQUIV="Set-Cookie"CONTENT="<!--#echo var="QUERY_STRING"-->;">
```

вставляємо рядок запиту, надісланий методом *GET* для установлення *cookie*.

Хід роботи

Завдання 1

1. При першому відкритті сторінки, вона повинна надіслати клієнтові *cookie*.
2. При наступних запитах сторінки, клієнт повинен посилати *cookie* серверу.
3. Сервер, отримавши це *cookie*, повинен видавати привітання, або повідомляти, що не знає такого клієнта.
4. Перевірте роботоздатність на сервері.
5. Вкажіть, у вигляді коментарів, для чого призначена кожна *SSI* команда.

Завдання 2

1. Створіть сторінку, за допомогою якої можна "зареєструвати" себе:
   - новачком;
   - користувачем;
   - досвідченим користувачем.
2. Оформіть реєстрацію, застосувавши форми (*radio* або *select*). Дані повинні відправлятися на другу сторінку методом *GET*.
3. На другій сторінці зробіть вставку змінної і її значення в тег *<META>* для установлення *cookie*.
4. Зробіть, на цій сторінці, обробку посланих *cookies* за допомогою директиви *IF* (із *SSI*) так, що б сторінка видавала, хто зайшов на сайт (новачок, користувач або досвідчений користувач).

5. Перевірте роботоздатність на сервері.

6. Вкажіть, у вигляді коментарів, для чого призначена кожна *SSI* команда.

Завдання 3

1. Створіть сторінку, з допомогою якою можна "обнулити" ваші *cookies*, тобто щоб при запиті цієї сторінки передавалися ваші *cookies* з простроченою датою.

2. Посилання на цю сторінку оформіте за допомогою кнопки "*submit*", застосувавши форми.

3. Перевірте роботоздатність на сервері.

До здачі лабораторної надаються: працюючі сторінки на сервері з коментарями вказаних тегів та їх властивості у вихідному файлі.

Зміст звіту

Ґрунтуючись на теоретичних відомостях та методичних вказівках до виконання експериментальної частини розв'язати з повним описом завдання.

Скласти звіт по роботі.

Контрольні запитання

1. Що таке належний рівень безпеки?

2. Якою мірою стандартні мережеві засоби, які встановлюються разом з операційній системою Windows, забезпечують належний рівень безпеки?

3. Які основні види порушення режиму мережевої безпеки Ви знаєте?

4. Для чого призначені маркери *cookie*? Який правовий режим маркерів *cookie*?

5. Навіщо творці веб-сторінок вбудовують в них активні об'єкти і активні сценарії? Яку загрозу вони можуть нести?

6. Що може послужити джерелом для збору відомостей про клієнтів мережі?

7. Як забезпечується захист від основних видів порушення режиму мережевої безпеки?

8. Як завжди забезпечується режим безпеки на державних підприємствах і в комерційних структурах?

9.4 Лабораторна робота №4 – Захист *IP*

Мета роботи. Вивчити архітектуру захисту на рівні комунікаційних протоколів. Навчитися реалізовувати та захищатися від флуд-атак.

Хід роботи

UDP Flood можна провести, використавши програму (з папки *Дисципліни/4 курс/Захист комп'ютерних мереж*) *udpflood.exe. ICMP Flood – icmp.exe.*

1. Запустіть програму *icmp.exe* (див. рис. 9.2):

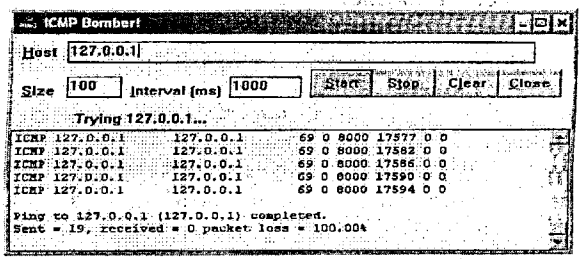


Рисунок 9.2 – Головне вікно запущеної програми ICMP

2. В поле "*Host*" введіть *IP*-адресу комп'ютера, який (за вказівкою викладача) можна атакувати безперервним потоком *ICMP*-пакетів, в поле *Size* – розмір кадру, *Intervsl* – частоту відсилання флуд-пакетів.

3. Далі проведіть такі ж дії на інших комп'ютерах лабораторії і одночасно натисніть *Start* для початку атаки.

4. Для програми *udpflood.exe* виконайте такі ж самі дії (див. рис. 9.3).

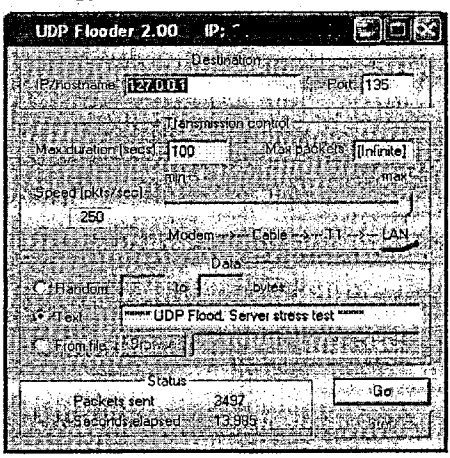


Рисунок 9.3 – Головне вікно запущеної програми UDP

5. В поле *IP/hostname* введіть адресу комп'ютера для атаки, *Port* – *135* і бігунець *Speed* поставте в крайню праву позицію (для максимальної швидкодії флуду). Натисніть *Go*.

6. Перегляньте вміст вікна *Быстродействие* операційної системи (на комп'ютері, що атакується, після початку атаки) і оцініть рівень падіння системних ресурсів під час проведення масованого флуду (натисніть *Ctrl + Alt + Del*).

Підготовка до роботи

Опираючись на матеріал теоретичних відомостей розділу 3, вивчити архітектуру захисту на рівні *IP* та призначення й основні можливості протоколу *IPSec*.

Завдання з експериментальної частини

1. Виконайте масовану атаку на вказаний викладачем комп'ютер-жертву.
2. Перегляньте на ньому системні характеристики швидкодії (завантаження процесора, використання файла підкачки, мережевого каналу).
3. Зробіть висновки.

Зміст звіту

1. Ґрунтуючись на теоретичних відомостях наведених в розділі 3 та методичних вказівках до виконання експериментальної розв'язати з повним описом завдання.
2. Скласти звіт з роботи.

Контрольні запитання

1. Які атаки можуть бути реалізовані на рівні протоколу *IP*?
2. Що таке флуд-атака. Наведіть приклади флуд-атак.
3. Обґрунтуйте необхідність введення протоколу *IPSec*.
4. Від яких атак захищає *IPSec*?
5. Які механізми захисту застосовуються в протоколі *IPSec*?
6. Що таке тунельний та транспортний режими в протоколі *IPSec*?

9.5 Лабораторна робота №5 – Захист на рівні протоколів *TLS/SSL*

Мета роботи. Навчитися отримувати сертифікати та забезпечувати захист засобами *SSL* веб-сайта на сервері *IIS (Internet Information Server)*, установлення *Certificate Services*, створення сертифіката, додавання центра сертифікації (*Certificate Authority*) в список довірених.

Хід роботи

Завдання 1. Створіть каталог та в ньому файл *default.htm*.

Завдання 2. Встановіть в себе на комп'ютері *Certificate Services*.

1. Після створення віртуального каталогу відкрийте в панелі управління консоль *Установка и удаление программ* і натисніть на кнопку *Установка компонентов Windows*.

2. У списку компонентів встановіть прапорець навпроти *Certificate Services*, натисніть *OK* , а потім натисніть на *Далі*.

3. На екрані *CA Type* виберіть *Stand-alone root CA*.

4. На екрані *CA Identifying Information* в полі *Common name for this CA* введіть ім'я вашого комп'ютера, решту значень залиште без змін. На решті екранів залиште значення за замовчуванням. У відповідь на попередження про необхідність рестарту *IIS* і включення *ASP* натисніть *OK*.

Завдання 3. Отримайте за допомогою *Certificate Services* цифровий сертифікат.

1. Після закінчення установлення *Certificate Services* відкрийте *Internet Explorer* і в адресному рядку введіть http://localhost/certsrv, а потім натисніть на **[Enter]**. Відкриється сторінка *Microsoft Certificate Services*. Натисніть на посилання *Request a certificate*.

2. На сторінці *Request a certificate* виберіть посилання *Advanced Certificate Request*.

3. На сторінці *Advanced Certificate Request* виберіть посилання *Create and Submit a request to this CA*.

4. На наступному екрані заповніть всі поля в групі *Identifying Information* (значення можна придумати). У поле *Name* обов'язково введіть ім'я вашого комп'ютера. У полі *Type of Certificate Needed* виберіть *Server Authentification Certificate*, в розділі *Key Options* встановіть прапорець *Store Certificate in the local computer certificate store*. Залиште в решті полів значення за замовчуванням і натисніть на кнопку *Submit*, потім натисніть на *Yes* у вікні підтвердження.

5. Відкрийте консоль *Certification Authority в Administrative Tools*. Розкрийте в цій консолі вузол комп'ютера, потім вузол *Pending Requests*, оберіть ваш запит, в контекстному меню оберіть *All Tasks -> Issue*.

6. Розкрийте вузол *Issued Certificates*, двічі клацніть по ньому правою кнопкою миші і проглянете інформацію про випущений вами сертифікат.

7. Поверніться в *Internet Explorer* на адресу http://localhost/certsrv і натисніть на посилання *View a status of a pending certificate request*. На наступній сторінці клацніть по рядку вашого запиту, і на сторінці *Certificate Issued* клацніть на посилання *Install this Certificate*, а потім натисніть на *Yes* у вікні підтвердження. Після появи повідомлення про те, що сертифікат успішно встановлений, закрийте вікна *Internet Explorer* і *Certification Authority*.

Завдання 4. За допомогою цього цифрового сертифіката захистіть створений вами віртуальний каталог так, щоб весь трафік при зверненні до цього віртуального каталога шифрувався за допомогою *SSL*.

1. Відкрийте *IIS Manager* і розкрийте в ньому вузол вашого комп'ютера -> *Web Sites*. Клацніть правою кнопкою миші по об'єкту *Default Web Site* (не віртуального каталога!) і в контекстному меню виберіть *Properties*. Перейдіть на вкладку *Directory Security* і натисніть на кнопку *Server Certificate*. Відкриється *Web Server Certificate Wizard*. На першому його екрані натисніть *Next*.

2. На екрані *Server Certificate* встановіть перемикач в положення *Assign an existing certificate*.

3. На екрані *Available Certificates* виберіть отриманий вами сертифікат.

4. На екрані *SSL Port* залиште порт за замовчуванням (443). Далі натисніть *Next*, а потім – *Finish*. Потім на вкладці *Directory Security* натисніть на кнопку *OK*.

5. Відкрийте властивості віртуального каталога і перейдіть на вкладку *Directory Security*. У групі *Secure Communications* натисніть на кнопку *Edit* і у вікні *Secure Communications* встановіть прапорець *Require secure channel* (*SSL*) (не встановлюйте прапорець *Require 128-bit encryption*). Натисніть двічі кнопку *OK*.

6. У вікні *Internet Explorer* звернетеся за адресою http://ім'я_вашого_компютера/ім'я_каталога. Повинно з'явиться повідомлення про помилку 403.4 *Forbidden – SSL is required*.

7. У вікні *Internet Explorer* зверніться за адресою https://ім'я_вашого_компютера/ім'я_каталога, а потім натисніть на *OK* у вікні підтвердження. Відкриється створений вами текстовий файл. Клацніть двічі мишею по іконці із замком в правій нижній частині екрана і проглянете інформацію про використовуваний сертифікат.

Завдання 5. Зверніться на свій віртуальний каталог з іншого комп'ютера і переконайтеся, що весь трафік дійсно шифрується.

1. Виконайте команду *Ping* на *IP*-адресу мережевого інтерфейсу вашого комп'ютера, щоб переконатися, що між комп'ютерами є мережеве з'єднання.

2. На другому комп'ютері запустіть будь-який сніфер та запустіть режим *Capture*.

3. На вашому комп'ютері відкрийте *Internet Explorer* і зверніться за адресою https://імя_вашого_компютера/імя_каталога. Відкриється вікно підтвердження, в якому буде повідомлено, що виданий сертифікат *CA* не знаходиться в списку довірених. Натисніть на *Yes*, щоб дістати доступ до вашої сторінки.

4. У сніфері зупиніть захват та прогляньте захоплені пакети.

Завдання 6. При зверненні з іншого комп'ютера на ваш веб-сайт з'являється застережливе повідомлення про неможливість перевірки сертифіката. Зробіть так, щоб це повідомлення більше не з'являлося.

1. На другому комп'ютері запустіть консоль *Certification Authority*, і в контекстному меню виберіть *Properties*.

2. На вкладці *General* виберіть єдиний сертифікат самого *CA* і натисніть на кнопку *View Certificate*.

3. Перейдіть на вкладку *Details* властивостей сертифіката і натисніть на кнопку *Copy to file*. Відкриється вікно майстра експорту сертифікатів. На його першому екрані натисніть на кнопку *Next*.

4. На екрані *Export File Format* залиште значення за замовчуванням (*DER encoded binary X.509*) і натисніть *Next*.

5. На екрані *File to Export* введіть повний шлях до створюваного файла. На наступному екрані натисніть на кнопку *Finish*, а потім *OK* – у вікні результатів експорту.

6. Скопіюйте файл сертифіката на комп'ютер.

7. На комп'ютері відкрийте *Internet Explorer*, в меню *Сервис* виберіть *Свойства обозревателя* і перейдіть на вкладку *Содержание*. Натисніть на кнопку *Сертификаты...* і перейдіть на вкладку *Доверенные корневые центры сертификации*. Натисніть на кнопку *Импорт*. Відкриється майстер імпорту сертифікатів. На його першому екрані натисніть на кнопку *Дальше*.

8. На екрані *Импортируемый файл* натисніть на кнопку *Обзор*, в списку *Тип файлов* виберіть *Сертіфікат X.509* і знайдіть скопійований вами файл сертифіката. Натисніть на кнопку *Открыть*, а потім – *Дальше*.

9. На екрані *Выберите хранилище сертификатов* залиште перемикач в положенні *Автоматически выбрать хранилище на основе типа сертификата* і натисніть *Дальше*, а потім – *Готовый*. Закрийте *Internet Explorer*.

10. На комп'ютері завершіть сеанс поточного користувача (щоб позбавитися всіх кешованих даних), увійдіть заново і ще раз зверніться в *Internet Explorer* за адресою https://імя_вашого_компютера/імя_каталога.

Жодних попереджень про недовір'я *CA* тепер видаватися не буде.

Зміст звіту

1. Ґрунтуючись на теоретичних відомостях розділу 4 та методичних вказівках до виконання експериментальної частини розв'язати з повним

описом завдання.

2. Скласти звіт з роботи.

Контрольні запитання

1. Для яких цілей може бути використаний сертифікат, випущений за шаблоном *User*?

2. Які параметри цифрового сертифіката можуть бути визначені за допомогою конфігураційного файла *CAPolicy.inf*?

3. Де розміщуються файли *CAPolicy.inf* і *Policy.inf*?

4. Для чого використовуються доповнення *AIA* і *CDP* в сертифікатах?

5. Які існують можливості для визначення точок доступу для сертифіката центра сертифікації та *CRL*?

6. Де настроюються доповнення *AIA* і *CDP* для сертифікатів, що випускаються даним центром сертифікації?

7. Які існують додаткові можливості в центрі сертифікації рівня підприємства в порівнянні з автономним центром сертифікації?

8. Які дії необхідно виконати для налаштування *SSL*-з'єднання між користувачем *user01* і корпоративним веб-сервером?

9.6 Лабораторна робота №6 – Захист поштових повідомлень

Мета роботи. Навчитися використовувати засоби для захисту електронної почти. Вивчити та засвоїти основні механізми безпеки, які застосовуються при побудові таких засобів.

Хід роботи

Завдання 1. Встановіть програмне забезпечення *PGP Desktop v9.0.6.*

1. Запустіть інсталяційний файл.
2. Виберіть мову.
3. Введіть ім'я користувача (*Name*), назву компанії (*Company*) і адресу електронної пошти;
4. Введіть ліцензійний ключ або оберіть один із інших варіантів.

Завдання 2. Створіть пару ключів

Для того, щоб мати можливість обмінюватися зашифрованими повідомленнями необхідно згенерувати пару ключів: відкритий і особистий. Ця пара буде використана надалі і для створення цифрового підпису. Не забувайте, що відкритим ключем ви можете ділитися з ким завгодно, а особистий ключ повинні знати лише Ви.

Послідовність дій для створення пари ключів.

1. Виконайте команду меню *File > New PGP Key.*
2. У діалоговому вікні, яке з'явилось, виконайте запропоновані інструкції.
3. Введіть повне ім'я та адресу електронної пошти;
4. Натисніть кнопку *„Advansed...”*, з'явиться вікно, в якому можна обрати властивості ключів, які будуть створені: тип (*RSA, Diffie-Hellman/DSS*), розмір (*1024-4096*), дату закінчення дії, алгоритм (*AES, CAST, TripleDES, IDEA, Twofish*), хеш-функцію (*MD-5, SHA-2-256, SHA-2-384, SHA-2-512, SHA-1, RIPEMD-160*);
5. Далі буде запропоновано ввести ключову фразу для створення пари ключів обміну.

Вкажіть власне ім'я (*Full name*) і адресу електронної пошти (*Email address*), не забуваючи, що саме ці дані будуть асоційовані програмою з Вашими ключами.

Виберіть тип ключа (*Key Pair Type*): ключ *RSA* повільніший за *Diffie-Hellman/DSS*, проте якщо серед ваших кореспондентів є користувачі версій нижче ніж *PGP 5.0*, доведеться використовувати ключ *RSA*.

Вибір довжини відкритого ключа (*Key Pair Size*): за замовчуванням, при використанні методу *Diffie-Hellman/DSS*, пропонується вибрати 2048-розрядний ключ.

Встановіть "дату смерті" ключової пари, тобто крайній термін, до якого дані ключі можуть бути використані для кодування і підпису (здатність до розшифровки і перевірки достовірності зберігатиметься).

Введіть ключову фразу (*passphrase*); існує єдине обмеження – у фразі повинно бути не менше 8 символів; можна використовувати будь-які регістри, спеціальні символи, пропуски, допускається запис фрази будь-якою мовою; якщо вас бентежить друк всліпу приберіть галочку *Hide Typing* (приховати надруковане), в цьому випадку текст відображатиметься "у відкриту".

Після чергового натиснення на кнопку "*Далі*" (*Next*) почнеться процедура генерації ключів.

6. Коли ключі згенеровані, вийдіть в *Інтернет* і відправте відкритий ключ на сервер (*root server – certserver.pgp.com*); до цієї процедури можна повернутися пізніше або віддати перевагу індивідуальній розсилці, яка, до речі, набагато більше гарантує достовірність вашого відкритого ключа.

7. Користувач, з яким ведеться листування, повинен знати відкритий ключ. Він може одержати його з сервера або з електронного листа.

8. Після цього користувач, який одержав ключ, зможе шифрувати листи направлені вам. Щоб посилати зашифровані листи йому, вам потрібно буде одержати його відкритий ключ. Підписувати листи ви можете і без відправки свого відкритого ключа іншим користувачам, але тоді ніхто не зможе перевірити ваш підпис. Ви також можете відправити свій відкритий ключ на публічно доступний сервер ключів, з якого цей ключ зможуть одержати інші користувачі.

9. Щоб вказати ступінь довіри, з яким ви відноситеся до того або іншого ключа, виділіть його і клацніть на ньому правою кнопкою миші, виберіть з контекстного меню пункт "*Properties –> Show Signing Key Properties*". Якщо ви встановите ступінь довіри до цього ключа ("*Trusted*"), інші ключі, підписані його власником, вважатимуться дійсними.

10. Щоб відкликати ключ, виділіть його і виберіть пункт "*Revoke*" з меню "*Keys*".

Завдання 3. Додайте цифровий підпис до повідомлення

Цифровий підпис підтверджує авторство і забезпечує цілісність листа або файла. Для цифрового підпису ви використовуєте свій закритий ключ, а перевірити авторство документа може хто завгодно за допомогою вашого відкритого ключа. Ступінь захисту цифрового підпису від підробки така ж висока, як ступінь стійкості при шифруванні за допомогою PGP.

1. Підпишіть документ, що зберігається у файлі *MS Word*. Знаходячись у файловому менеджері, клацніть по файлу правою кнопкою миші і знайдіть в меню пункт *PGP – Sign*.

2. Введіть пароль для закритого ключа. Подивіться: поряд з файлом з'являється маленький файл з розширенням *.sig* (англ. *signature* – "підпис").

3. Клацніть по файлу *sig*. З'явиться віконце, в якому вказано, як називається файл-документ, а також хто і коли його підписав. Тепер відкрийте документ в редакторі *Word* і зробіть яку-небудь маленьку зміну – наприклад, додайте пропуск в кінці абзацу. На око ця правка абсолютно

непомітна. Збережіть документ, закрийте його і знову клацніть по файлу підпису *sig*. Там, де була вказана дата підпису, тепер напис *Bad signature* (неправильний підпис).

Завдання 5. Верифікуйте повідомлення, забезпечене цифровим підписом.

Для верифікації одержаного підписаного повідомлення необхідно вибрати команду головного меню "*Tools -> Verify Files*".

Алгоритм обробки цифрового підпису дозволяє з'ясувати, чи є відправник повідомлення, вказаний в заголовку листа, і той, хто поставив цифровий підпис, однією і тією ж особою, і чи не зазнало повідомлення модифікацію в дорозі.

Завдання 6. Зашифруйте повідомлення через буфер обміну.

Для зашифровування будь-якого повідомлення.

1. Виділіть текст (наприклад, в редакторі *Outlook Express* або в *Microsoft Word*) і перемістіть (скопіюйте) його в буфер пам'яті (*Clipboard*).

2. Клацніть по зображенню *PGPTray* в правому нижньому кутку *Windows* і виберіть в меню *Clipboard – Encrypt* (зашифрувати вміст буфера обміну).

3. У вікні відобразяться всі доступні відкриті ключі.

4. Двічі клацніть мишею по потрібному ключу (або ключах, якщо ви шифруєте відразу для декількох адресатів) у верхньому віконці. Ключ (або ключі) переміститься в нижнє віконце. Натисніть *OK*.

5. Тепер в буфері обміну знаходиться вже не оригінальний, а зашифрований текст. Можете вставити його в редакторі своєї поштової програми командами меню або поєднанням клавіш [*Ctrl*]+[*V*].

6. Оскільки в зашифрованому тексті використані тільки текстові символи, його можна без проблем відправляти по електронній пошті "як є".

Іноді буває корисно зашифрувати повідомлення, крім відкритого ключа адресата, ще і своїм власним відкритим ключем. Тоді, у разі чого, ви зможете одержати до нього доступ.

Для розшифровки повідомлення PGP через буфер обміну.

1. Скопіюйте потрібний фрагмент в буфер.

2. Викличте *PGPtray* і виберіть в меню *Clipboard – Decrypt and Verify*.

3. З'явиться вікно для введення пароля. Пароль потрібен для доступу до вашого закритого ключа. Введіть пароль і натисніть *OK*.

4. З'явиться невелике віконце перегляду тексту (*Text Viewer*), і в ньому ваш розшифрований текст. Скопіюйте його в буфер обміну ([*Ctrl*]+[*C*]) і далі робіть з ним те, що захочете.

Завдання 7. Зашифруйте файл

Досить часто виникає ситуація, коли потрібно відправити поштою файл-додаток до листа – наприклад, документ *Word* або таблицю *Excel*. Буває, що окремі файли на диску потребують захисту. За допомогою *PGP*

можна успішно шифрувати і файли. Найпростіший шлях: у файловому менеджері виберіть потрібний файл, клацніть правою кнопкою миші і в меню знайдіть пункт *PGP –> Encrypt*. Вам залишилося вибрати відкритий ключ. Зашифрований файл з'явиться в тій же теці, що і оригінал, але з розширенням *.pgp*. Можна шифрувати декілька файлів і навіть теки з файлами.

Для шифрування даних на диску, щоб потім мати до них доступ, використайте власний відкритий ключ.

Розшифровувати файли ще простіше, ніж текст. Потрібно двічі швидко клацнути лівою кнопкою миші по зашифрованому файлу, немов ви запускаєте програму.

Зміст звіту

Користуючись теоретичними відомостями, розв'язати з повним описом завдання (завдання вказує викладач).

Скласти звіт з роботи.

Контрольні запитання

1. Як потрібно передавати відкриті ключі *PGP* своїм кореспондентам?
2. Що таке компрометація цифрового підпису?
3. Що впливає на криптостійкість цифрового підпису?
4. Який ключ використовується при шифруванні повідомлення?
5. Який ключ використовується при створенні цифрового підпису?
6. Як відправляти файли на захищене зберігання за допомогою програми *PGP*?
7. За допомогою ще яких програм можна забезпечити захищене зберігання файлів?
8. Чи існує вірогідність втрати (видалення) захищених файлів?
9. Що необхідно зробити для виключення випадкового або навмисного видалення файлів?
10. Для чого використовується і які особливості електронного цифрового підпису?

9.7 Лабораторна робота №7 – Віддалений доступ до комп'ютера

Мета роботи. Навчитися використовувати засоби для отримання віддаленого доступу та віддаленого керування робочими станціями у локальній мережі.

Підготовка до роботи

Anyplace Control™ – програма для керування віддаленим комп'ютером через локальну або глобальну мережу, яка відображає екран іншого комп'ютера на моніторі. Програма дозволяє дистанційно керувати віддаленим комп'ютером за допомогою локальних миші та клавіатури. Тобто надає можливість працювати за віддаленим комп'ютером так, наче за ним сидите. Програма дозволяє працювати з декількома віддаленими комп'ютерами одночасно, з будь-якої точки світу.

Можливості:

- відображення віддаленого робочого столу на екрані;
- управління віддаленим комп'ютером за допомогою клавіатури та мишки;
- обмін різними даними між комп'ютерами;
- дистанційне виключення комп'ютера, перезавантаження і т.д.

Можливі області застосування *Anyplace Control.*

- **Управління мережею**: *Anyplace Control* дозволяє адміністратору мережі одержати повний контроль над будь-яким мережевим комп'ютером дистанційно, без необхідності фізичної присутності поблизу віддаленого комп'ютера.
- **Дистанційна технічна підтримка, віддалена допомога**: *Anyplace Control* дозволяє відділу технічної підтримки віддалено управляти комп'ютерами користувачів і швидко усувати технічні неполадки. Це оптимальне рішення для усунення комп'ютерних проблем дистанційно.
- **Використовування домашніх комп'ютерних мереж**: Використовуючи *Anyplace Control*, Ви можете допомогти своїм родичам і друзям усунути деякі комп'ютерні проблеми, не виходячи зі своєї квартири.

Хід роботи

1. На бригаду повинно бути два (або більше) комп'ютери, підключених до локальної мережі або *Інтернет.*

2. Встановіть програмне забезпечення на обох комп'ютерах. Програма складається з двох модулів.

Модуль *Admin* (інша назва – додаток-клієнт) відображає віддалений робочий стіл на екрані сервера (у окремому вікні) і дозволяє вам управляти віддаленим комп'ютером мишею і клавіатурою. Звичайно цей модуль встановлюється тільки на комп'ютер адміністратора.

Модуль *Host* (інша назва – додаток-сервер) виконує команди надіслані *Admin*-модулем через мережу. Цей модуль встановлюється на другому комп'ютері, який буде управлятися дистанційно (як правило, це мережевий комп'ютер).

Інсталяційний пакет містить обидва модулі. Ви можете вибрати необхідний модуль під час установлення програми. Також можна встановити обидва модулі одночасно на той же самий комп'ютер.

3. Запустіть *Admin*-модуль на вашому комп'ютері (за допомогою ярлика на робочому столі або через меню *Пуск ->Програми -> Anyplace Control).*

4. Додайте *IP*-адресу (або мережеве ім'я) віддаленого комп'ютера в адресну книгу.

5. Підключіться до віддаленого комп'ютера, натиснувши кнопку "*Подключить*".

6. Натисніть кнопку "*Управление*" та керуйте комп'ютером дистанційно за допомогою своєї миші і клавіатури у вікні, що з'явилося, із зображенням віддаленого робочого столу.

Зміст звіту

Користуючись теоретичними відомостями, розв'язати з повним описом завдання (завдання вказує викладач).

Скласти звіт з роботи.

Контрольні запитання

1. Що таке віддалений доступ до комп'ютера? За допомогою яких механізмів можна його реалізувати?

2. Які засоби віддаленого доступу до комп'ютера Ви знаєте? Які використовували?

3. За рахунок яких механізмів операційних систем можливий віддалений доступ до комп'ютера?

4. Яким чином можна захиститися від проникнення до комп'ютера через механізми віддаленого доступу?

5. Яким чином можна організувати віртуальну приватну мережу?

9.8 Лабораторна робота № 8 – Основні ознаки присутності на комп'ютері шкідливих програм

Мета роботи. Навчитися виявляти шкідливе програмне забезпечення, яке з'явилося на робочій станції, з використанням як явних, опосередкованих, так і прихованих ознак.

Хід роботи

Перед початком лабораторної роботи переконайтеся, вхід у систему виконано під обліковим записом, що має права адміністратора.

Завдання 1. Вивчити налаштовування браузера.

У цьому завданні пропонується досліджувати явні прояви вірусної активності на прикладі несанкціонованої зміни настроювань браузера.

1. Відкрийте браузер *Internet Explorer*, скориставшись однойменним ярликом на робочому столі або в системному меню *Пуск/Программы*.

Повинна відкритися сторінка за замовчуванням.

2. Перевірте значення параметрів, відповідальних за стартову сторінку. Для цього потрібно скористатися меню *Сервис*. Відкрийте його й виберіть пункт *Свойства обозревателя*.

3. Адреса стартової сторінки зазначена у першому ж полі вікна *Свойства обозревателя*, на закладці *Общие*.

4. Змініть це поле, увівши адресу www.vstu.edu.ua.

5. Закрийте й знову відкрийте браузер. Переконаєтеся, що тепер першою була завантажена сторінка www.vstu.edu.ua.

Таким чином, якщо браузер почав самостійно завантажувати сторонній сайт, у першу чергу потрібно вивчити настроювання браузера: яка адреса виставлена в полі домашньої сторінки.

Ряд шкідливих програм обмежуються зміною цього параметра й для усунення наслідків зараження потрібно лише виправити адресу домашньої сторінки. Однак це може бути тільки частиною шкідливого навантаження. Тому якщо Ви виявили несанкціоновану зміну адреси домашньої сторінки, варто негайно встановити антивірусне програмне забезпечення й перевірити весь жорсткий диск на наявність вірусів.

Завдання 2. Визначення підозрілих процесів.

Одним з основних проявів шкідливих програм є наявність у списку запущених процесів підозрілих програм. Це часто допомагає при виявленні шкідливих програм, що мають лише приховані або непрямі прояви.

Однак необхідно чітко розуміти й уміти відрізняти легальні процеси від підозрілих.

1. Перейдіть до вікна *Диспетчер задач Windows*, натиснувши одночасно клавіші **[Ctrl]**, **[Shift]** та **[Esc]**.

2. Відкрите вікно містить чотири закладки, що відповідають чотирьом видам активності, які відслідковує *Диспетчер*: *Приложения*,

Процессы, *Быстродействие* (використання системних ресурсів), *Сеть* та *Пользователи*. За замовчуванням повинна відкритися друга закладка, *Процессы*.

3. Уважно вивчіть поданий у вікні список процесів. Якщо на комп'ютері не запущені жодні користувацькі програми, він повинен містити тільки службові процеси операційної системи. Опис більшості процесів можна знайти в *Інтернет*.

4. Для кожного процесу виводяться його параметри: ім'я образа (може не збігатися з ім'ям файла, що його запускає), ім'я користувача, від імені якого був запущений процес, завантаження цим процесом процесора та обсяг займаної ним оперативної пам'яті.

5. Відсортуйте всі процеси за використанням ресурсів процесора. Для цього натисніть на заголовок поля *ЦП*. Оскільки в цей момент не повинна бути запущена жодна користувацька програма, процесор повинен бути вільний.

6. У ряді випадків може знадобитися вручну завершити якийсь процес. Це можна зробити за допомогою кнопки *Завершить процесс* у контекстному меню.

7. Випишіть всі запущені процеси на аркуш паперу або в текстовий файл і перейдіть до закладки *Приложения*. Оскільки в цей момент не запущений жодний додаток, список запущених додатків порожній.

8. Не закриваючи вікна *Диспетчер задач Windows*, відкрийте програму *Paint*.

9. Не закриваючи додаток *Paint*, поверніться до вікна *Диспетчер задач Windows* і простежте за змінами на закладці *Приложения*. Список запущених додатків повинен містити рядок, який відповідає *Paint*. Оскільки він зараз працює, це ж записано в рядку *Состояние*.

10. Перейдіть до закладки *Процессы*. Порівняєте список запущених зараз процесів з переліком, складеним у п. 5 цього завдання. Знайдіть відмінність. З'ясуйте, який процес відповідає програмі *Paint*.

11. Перейдіть до закладки *Быстродействие*.

12. Уважно вивчіть розташовані тут графіки. Будь-які сплески на них повинні за часом відповідати якимсь діям, наприклад, запуску вимогливої до ресурсів програми. Якщо нічого схожого свідомо не запускалося, це може бути причиною для більш детального дослідження комп'ютера

13. Закрийте вікно *Диспетчер задач Windows*.

Завдання 3. Дослідити елементи автозапуску

Для того щоб прикладна програма почала виконуватися, її потрібно запустити. Отже, і вірус має потребу в тому, щоб його запустили. Для цього можна використати два сценарії: або зробити так, щоб користувач сам його стартував (використовуючи обманні методи), або залізти в конфігураційні файли й запускати одночасно з іншою, корисною програмою. Оптимальним з погляду вірусу варіантом служить запуск

одночасно з операційною системою – у цьому випадку запуск практично гарантований.

Найпростіший спосіб додати будь-яку програму в автозавантаження – це помістити її ярлик у розділ *Автозагрузка* системного меню *Пуск->Программы*. За замовчуванням, відразу після установлення операційної системи цей розділ порожній, оскільки ні однієї прикладної програми ще не встановлено.

1. Перевірте папку *Автозагрузка* на Вашому комп'ютері. Вона повинна бути порожня.

2. Додайте в список автозавантаження свою програму. Для цього двічі клацніть лівою клавішею миші за назвою групи *Автозагрузка*. Все, що потрібно зробити, щоб якась програма запускалася автоматично при старті операційної системи – це помістити в цю папку її ярлик.

3. Повторіть дії пункту 2, але тільки для папки *Пуск->Программы->Стандартные*. У вікні, що відкрилося, знайдіть ярлик *Блокнот*. Скопіюйте його до вікна *Автозагрузка*.

4. Закрийте вікно й переконаєтеся, що тепер розділ *Автозагрузка* в системному меню *Пуск->Программы* не порожній

5. Перезавантажте комп'ютер і ввійдіть у систему під своїм обліковим записом. Переконаєтеся, що після завершення завантаження автоматично запустилася програма *Блокнот*.

При обстеженні комп'ютера потрібно пам'ятати, що відсутність підозрілих ярликів у розділі *Автозагрузка* системного меню *Пуск->Программы* не гарантує, що жоден додаток не запускається автоматично. Технічно для автозавантаження потрібно додати відповідний запис до системного реєстру операційної системи.

6. Для більшості ситуацій, пов'язаних з автозавантаженням, досить використати системну утиліту *Настройка системы*.

7. Запустіть утиліту *Настройка системы*.

8. У вікні *Запуск программы* наберіть *msconfig* і натисніть *ОК*.

9. На першій закладці, *Общие*, можна вибрати варіант запуску операційної системи. За замовчуванням відзначений *Обычный запуск*. Він забезпечує максимальну функціональність системи. Інші два варіанти запуску призначені для діагностування.

Другий режим, *Диагностичный запуск*, рекомендується використати також при вірусному інциденті, що підтвердився, – якщо комп'ютер уже заражений, відразу встановити антивірус у ряді випадків не можна, наприклад, якщо вірус свідомо блокує запуск ряду антивірусних програм. Тоді, якщо немає можливості видалити або хоча б тимчасово знешкодити вірус вручну, рекомендується запустити операційну систему в безпечному режимі, установити антивірус і відразу ж перевірити весь жорсткий диск на наявність вірусів.

Для одержання додаткової інформації про цю закладку й інші можна

скористатися кнопкою *Справка*.

10. Перейдіть на закладку *Службы*. Кожна служба являє собою якийсь додаток, що працює у фоновому режимі. Однак зараз ніяких сторонніх служб, крім системних, установлено бути не повинно. Переконаєтеся в цьому, відзначивши прапорець: *Не отображать службы Microsoft*. Якщо сторонніх додатків дійсно немає, список повинен стати порожнім.

11. Перейдіть до останньої закладки, *Автозагрузка*, і переконаєтеся, які в списку додатків, що запускають автоматично при завантаженні системи, є *Блокнот*. Відзначимо, що список у вікні *Настройка системы* може містити додаткові елементи, не відображувані в розділі *Пуск ->Программы->Автозагрузка*.

12. Відключіть автоматичне завантаження *Блокнота*, очистивши прапорець у стовпці *Элемент автозагрузки* та натисніть *ОК*. У відкритому вікні дозвольте провести перезавантаження.

13. Дочекайтеся закінчення перезавантаження й увійдіть у систему під своїм обліковим записом.

14. Перейдіть до закладки *Автозагрузка* й переконаєтеся, що її вигляд не змінився – *Блокнот* так само присутній у списку, але відключений.

15. Перевірте, що *Блокнот* автоматично не запустився.

16. Поверніться до закладки *Общие* вікна *Настройка системы* й виберіть сценарій *Обычный запуск*. Натисніть *ОК* й у такому вікні виберіть *Перезагрузка*.

17. Дочекайтеся закінчення перезавантаження, увійдіть у систему під своїм обліковим записом і переконаєтеся, що повідомлення про вибірковий запуск не з'являється.

18. Однак оскільки прапорець, знятий на кроці 12, при перемиканні в режим *Обычный запуск* повернувся (*Обычный запуск* припускає завантаження всіх зареєстрованих компонентів), додаток *Блокнот* знову автоматично запускається після завершення перезавантаження операційної системи. Переконаєтеся, що в *Пуск->Программы->Автозагрузка* повернувся ярлик *Блокнота*.

19. Видаліть його. Тепер автозавантаження чисте. Переконайтеся в цьому, виконавши перезавантаження й увійшовши в систему під своїм обліковим записом.

Завдання 4. Дослідити мережеву активність.

Зненацька підвищена мережева активність може служити яскравим свідченням роботи на комп'ютері підозрілих програм, які роблять несанкціоноване розсилання листів, зв'язуються зі своїм автором і передають йому конфіденційну інформацію, просто завантажують свої додаткові модулі або атакують сусідній комп'ютер. Але при цьому потрібно не забувати, що ряд цілком легальних додатків також мають

властивість іноді зв'язуватися із сайтом фірми-виробника, наприклад, для перевірок наявності оновлень або більше нових версій.

1. Відкрийте вікно *Диспетчер задач Windows*, натиснувши одночасно клавіші **[Ctrl]**, **[Shift]** та **[Esc]**, і перейдіть до закладки *Сеть*. Оскільки зараз не ініціюється жодного мережевого з'єднання, графік повинен бути порожній, вірніше являти собою пряму на рівні 0 %.

2. Ініціюйте будь-яке мережеве з'єднання. Наприклад, відкрийте браузер і завантажте сайт www.vstu.edu.ua.

3. Простежте за змінами на графіку *Диспетчер задач*: всі Ваші дії відобразяться на графіку у вигляді піків мережевої активності, а значення поля *Использование сети* на деякий час перестане бути рівним нулю.

Диспетчер задач Windows показує тільки загальну інформацію. Для одержання більше докладних даних можна скористатися утилітою *netstat*.

4. Закрийте вікно *Диспетчер задач Windows* і перейдіть до системного меню *Пуск->Программы->Стандартные->Командный рядок*. Наберіть

```
netstat -a
```

і натисніть **[Enter]**.

Результатом виконання команди є список активних підключень, у який входять установлені з'єднання й відкриті порти.

Частина портів пов'язана із системними службами *Windows* і відображається за назвою. Порти, що не відносяться до стандартних служб, відображаються за номерами.

UDP-порти позначаються рядком *UDP* у колонку *Ім'я*. Вони не можуть перебувати в різних станах, тому спеціальна позначка *LISTENING* у їхньому відношенні не використовується. Як й *TCP*-порти вони можуть відображатися за іменами або номерами.

Порти, використовувані шкідливими програмами, найчастіше є нестандартними й тому відображаються відповідно до їхніх номерів. Втім, можуть зустрічатися троянські програми, що використовують для маскування стандартні для інших додатків порти, наприклад 80, 21, 443 – порти, використовувані на файлових та веб-серверах.

5. Перевірте, як зміниться статистика, відображувана *netstat* при ініціюванні нових з'єднань. Команда *netstat*, на відміну від *Диспетчер задач Windows*, не працює в режимі реального часу, а відображає миттєву статистику.

6. Досліджуйте отриману статистику. Закрийте браузер, повторіть команду `netstat -a` і натисніть **[Enter]**. Переконайтеся, що всі викликані раніше мережеві з'єднання закриті, а перелік активних з'єднань не відрізняється від даних

7. Закрийте вікно командного рядка. Для цього введіть команду

```
exit
```

і натисніть **[Enter]**.

8. Іноді зібрані дані дозволяють визначити ім'я вірусу, тоді можна

звернутися, наприклад, до вірусної енциклопедії www.viruslist.com, щоб вручну ліквідувати наслідки зараження. Якщо однозначної відповіді одержати не вдається, необхідно зібрати всі підозрілі прояви й звернутися до *Інтернет*. На сьогоднішній день існує досить багато сайтів, що містять описи безпечних процесів, наприклад www.processlibrary.com. Зрівнявши отримані в результаті аналізу дані із поданими в бібліотеці описами, потрібно залишити не заявлені як легальні процеси й об'єкти й простежити їхнє розташування на диску.

Подальші дії залежать від того, чи використовується на комп'ютері антивірусна програма чи ні. Якщо ні, то отримані файли потрібно досліджувати за допомогою антивірусної програми, наприклад онлайн сканера http://www.kaspersky.ru/virusscanner, що дозволяє безкоштовно перевіряти окремі об'єкти.

Якщо на комп'ютері антивірус уже встановлений, після виявлення підозрілих файлів варто звернутися в службу технічної підтримки антивірусної компанії, чий продукт використається на комп'ютері, прикріпивши до повідомлення виявлені підозрілі об'єкти. Цілком можливо вони містять новий, ще не відомий вірус.

Зміст звіту

Користуючись теоретичними відомостями з розділу 7, розв'язати з повним описом завдання (завдання вказує викладач).

Скласти звіт з роботи.

Контрольні запитання

1. Які види шкідливого програмного забезпечення Ви знаєте?
2. Що таке явні прояви шкідливого програмного забезпечення? Яким чином їх перевірити?
3. Перерахуйте непрямі ознаки наявності шкідливого програмного забезпечення. Яким чином їх перевірити?
4. В чому основна небезпека шкідливого програмного забезпечення з прихованими проявами? Як його виявити? За допомогою яких засобів?
5. Як перевірити процеси *Windows* та виділити з них підозрілі? Яким чином перевірити чи процес є легальним?
6. Що таке автозапуск? Навіщо його використовують шкідливі програми?
7. Перерахуйте всі місця де можна перевірити, чи знаходяться в автозапуску шкідливі програми.
8. Як перевірити мережеву активність? Перерахуйте всі відомі Вам засоби.

9.9 Лабораторна робота № 9 – Діагностика антивірусу Касперского

Мета роботи. Навчитися працювати з антивірусними програмами, тестувати їх придатність до роботи на робочій станції, виявляти шкідливе програмне забезпечення, лікувати тощо.

Хід роботи

Завдання 1. Створіть тестові віруси *EICAR*, *CURE-EICAR* та *SUSP-EICAR*.

1. Викличте контекстне меню іконки *Антивірусу Касперского* в системній панелі й виберіть пункт *Приостановка защиты*. Після цього з'явиться повідомлення про те, що захист не працює, а іконка *Антивірусу Касперского* знебарвиться.

2. Запустіть текстовий редактор *Блокнот*. У вікні наберіть

```
X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
```

3. Збережіть файл, що вийшов, під ім'ям *eicar.com* в *D:\Test*.

4. Модифікуйте *EICAR*, додавши до нього приставку *CURE-*. Збережіть файл під ім'ям *D:\Test\cure-eicar.com*.

5. Аналогічно створіть *SUSP-EICAR*, але для приставки *SUSP-*.

6. Закрийте вікно текстового редактора *Блокнот*.

7. У результаті цих дій у папці *D:\Test* повинно з'явитися три файли: *eicar.com*, *cure-eicar.com* й *susp-eicar.com*. Переконайтеся в цьому.

8. Перевірте розмір кожного з файлів. Файл *eicar.com* повинен мати розмір 68 байт, а *cure-eicar.com* й *susp-eicar.com* – по 73 байти.

9. Переконайтеся, що при запуску тестовий вірус виводить попереджувальне вікно. Для того, щоб побачити повідомлення про те, що *EICAR* – це тестовий вірус, потрібно запустити його через командний рядок.

10. Скористайтеся системним меню *Пуск->Програми-> Стандартні -> Командний рядок*.

11. Перейдіть до потрібного каталога, запустіть файл *eicar.com*, набравши команду `eicar.com` і натиснувши **[Enter]**. Ознайомтеся з повідомленням, що вивів *EICAR*. Закрийте вікно командного рядка.

Завдання 2. Протестуйте антивірус за допомогою *EICAR*.

1. Перейдіть до папки з тестовими вірусами.

2. Викличте контекстне меню файла *eicar.com* і виберіть пункт *Проверить на вирусы*. Як результат, майже одночасно повинні з'явитися два вікна. Спочатку вікно статистики виконання завдання пошуку вірусів. Поверх нього в лівому верхньому куту екрана – вікно із запитом дії.

Ви можете або пропустити, або видалити *eicar.com*.

3. Натисніть *Удалить*. *Антивірус Касперского* видалить *eicar.com*. Це відразу ж відобразиться у вікні статистики. Вивчіть подані в ньому дані й натисніть *Закрыть*.

Перевірте, що вилучений файл *eicar.com* з'явився в резервному сховищі. Для цього відкрийте головне вікно інтерфейсу, двічі клацніть на іконці *Антивірусу Касперского* в системній панелі.

4. Перейдіть до розділу *Сервис*, а потім до підрозділу *Файлы данных*. Зверніть увагу на зведену інформацію про резервне сховище, розташовану в групі *Резервное хранилище*. Тепер у ньому зберігається один файл і отже розмір резервного сховища відмінний від нуля. Перейдіть до вікна з докладною інформацією про резервне сховище, натиснувши на групу *Резервное хранилище*.

5. Виділіть рядок "*Зараженный: вирус EICAR-Test-File*". Як тільки був обраний об'єкт, стали активними кнопки керування ним: *Удалить* та *Восстановить*.

6. Видаліть *eicar.com*, натиснувши *Удалить*.

7. Переконайтеся, що резервне сховище тепер порожнє й закрийте вікно статистики, натиснувши *Закрыть*.

Завдання 3. Вилікуйте інфіковані файли.

1. У головному вікні інтерфейсу перейдіть до розділу *Защита*.

2. Включіть постійний захист, натиснувши *Пуск*. Переконаєтеся, що це відбулося, простеживши за змінами в зовнішньому вигляді вікна.

3. Відкрийте папку *D:\Test*.

4. Оскільки раніше була дана вказівка видалити файл *eicar.com*, у папці залишилося тільки два файли, *cure-eicar.com* й *susp-eicar.com*.

5. Зверніться до файла *cure-eicar.com*. *Антивірус Касперского* відразу ж повинен виявити, що Ви намагаєтеся відкрити заражений файл. Як результат – виводиться вікно із запитом дії, яку потрібно застосувати до знайденого вірусу.

6. Вилікуєте файл *cure-eicar.com*, натиснувши *Лечить*. Про успішний результат лікування повідомляє інформаційне вікно.

7. Перевірте, що залишений в папці *D:\Test* файл *cure-eicar.com* має розмір усього 4 байти. Переконайтеся, що перед лікуванням файл був поміщений у резервне сховище.

8. Очистіть резервне сховище, виділивши рядок з *EICAR* і натиснувши *Удалить*.

Завдання 4. Перемістіть файли на карантин.

На карантин ставляться всі підозрілі файли – тобто такі, які за всіма ознаками інфіковані, але вердикт про їхню невиліковність поки не винесений. Можливо при такому відновленні антивірусних баз у них буде додана інформація, що дозволяє це зробити, або ж засвідчить про невиліковність. У першому випадку можна буде провести повторну перевірку карантину й вилікувати файли, що піддаються лікуванню, у другому – видалити з переміщенням у резервне сховище.

Ще одна відмінність карантину від резервного сховища полягає в тому, що на карантин можна ставити об'єкти вручну, наприклад, якщо

вони в користувача все-таки викликають підозри, незважаючи на негативну відповідь антивірусу.

1. Спочатку перейдіть до папки *D:\Test* і натисніть один раз на іконку файла *susp-eicar.com*. *Антивірус Касперского* просканує цей файл і виявить його підозрілим. У результаті виведеться вікно із проханням вибрати необхідну дію. Вам буде запропоновано три варіанти: помістити на карантин, видалити або пропустити. Оскільки файл визнаний підозрілим, лікування неможливо (інакше б було запропоновано його спочатку вилікувати).

2. Натисніть *Карантин*. Зверніть увагу, що після приміщення *susp-eicar.com* на карантин, у папці *Test* залишився тільки вилікований у попередньому завданні *cure-eicar.com*.

3. Тепер простежте, що *susp-eicar.com* з'явився на карантині. Для цього відкрийте головне вікно антивірусу й перейдіть до підрозділу *Файлы данных*. Зверніть увагу на зведену статистику по карантині й клацніть на групі *Карантин*.

4. Виділіть рядок з *susp-eicar.com* і клацнувши на ньому правою клавішею миші, виведіть контекстне меню. Зверніть увагу на пункт *Отправить*. Якщо на комп'ютері встановлений поштовий агент, скориставшись цим пунктом можна швидко сформувати й відправити в службу технічної підтримки *Лабораторії Касперского* лист із проханням перевірити обраний файл.

5. Закрийте вікно статистики, натиснувши *Закрити*.

Зміст звіту

Користуючись теоретичними відомостями з розділу 7, розв'язати з повним описом завдання (завдання вказує викладач).

Скласти звіт з роботи.

Контрольні запитання

- 1. Які дії повинні бути зроблені при поява повідомлення "Лікування не можливе"?
- 2. Що означає перевірка програм за форматом?
- 3. Що означає перевірка програм за розширенням?
- 4. Як задати перевірку файлів за маскою?
- 5. Які Ваші дії, якщо програма видала повідомлення про підозру на зараження об'єкта вірусом?
- 6. Призначення програми підготовки дисків аварійного відновлення?
- 7. Яке призначення *Монітору*?
- 8. Що таке карантин?
- 9. Яка структура тестового вірусу?
- 10. Які види тестового вірусу є і які Ви тестували?

ЛІТЕРАТУРА

1. Азаров О. Д., Захарченко С. М. Основи побудови та адміністрування мережевих операційних систем. – Вінниця: ВДТУ, 2001. – 144 с.
2. Столингс В. Криптография и защита сетей: принципы и практика. Основы защиты сетей: Приложение.2-е изд. Пер. с англ. – М.: Издательский дом «Вильямс», 2001. – 672 с.
3. Касперски К. Техника сетевых атак. – М.: Солон-Р, 2001. – 524 с.
4. Хошаба О. М. Протоколи основних служб Internet – Вінниця: УНІВЕРСУМ-Вінниця, 2003. – 214 с.
5. Калита Д. М. Комп'ютерні мережі. Апаратні засоби та протоколи передачі даних, К.: «МК-Пресс», 2003. – 156 с.
6. Мамаев М., Петренко С. Технология защиты информации в интернете: Специальный справочник. – СПб.: Питер, 2002. – 848 с.
7. Бабак В. П., Корченко О. Г. Інформаційна безпека та сучасні мережеві технології, -2003. – 248 с.
8. Защита информации в телекоммуникационных системах/Г. Ф. Конахович, В. П. Климчук, С. М. Паук. – К.: «МК-Пресс», 2005. – 288 с.
9. Хаулет Т. Защитные средства с открытыми исходными текстами БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий - ИНТУИТ.ру, 2007 // http://www.intuit.ru/department.
10. Романец Ю. В., Тимофеев П. А., Шаньгин В. Ф. Защита информации в компьютерных системах и сетях // Под ред. В. Ф. Шаньгина. - М.: Радио и связь, 2001. - 376 с.
11. Бабаш А. В., Шанкин Г. П. Криптография. Под ред. В. П. Шерстюка, Э. А. Применко/ А. В. Бабаш, Г. П. Шанкин. - М.: СОЛОН-Р, 2002. - 512 с.
12. Анонимный автор. Максимальная безопасность в Linux: Руководство по защите серверов и рабочих станций Linux, написанное зловмисником. – К.: «ДиаСофт», 2004 – 400 с.
13. Матеріали з мережі INTERNET.

ГЛОСАРІЙ

AES	*Advanced Encryption Standard* Покращений стандарт шифрування Стандарт *AES* прийнятий в 2000 р. в США замість стандарту *DES*. Стандарт симетричного блокового шифрування (довжина блоку – 128 біт). *AES* підтримує 128-розрядні ключі, але може і 192- та 256-розрядні. Базується на алгоритмі *Rijndael*, який розроблений криптограмами *Joen Daemer* та *Vincent Rijmen* Офіційний сайт - http://www.csrc.nist.gov/encryptions/aes/
AH	В протоколі IPSec *Authentication Header* Заголовок автентифікації Протокол автентифікації забезпечує підтримку цілісності даних і автентифікацію пакетів *IP*. Перше гарантує неможливість таємної модифікації вмісту пакета на шляху проходження. Функція автентифікації дозволяє кінцевій системі або мережевому пристрою ідентифікувати користувача або програмний додаток і, відповідно, відфільтрувати трафік та захиститися від атак з підміною мережевих адрес та атак відтворення повідомлень
ANSI	*American National Standards Institute* Американський національний інститут стандартів Об'єднання американських промислових і ділових груп, яке розробляє торгові і комунікаційні стандарти, член *ISO*. Сформований 19 жовтня 1918. В *ANSI* представлені американські корпорації, урядові служби, міжнародні організації і приватні особи. Офіційний сайт ansi.org
API	*Application Programing Interface* Інтерфейс прикладного програмування Набір стандартних програмних переривань, викликів процедур (методів) та форматів даних, які повинні використовувати прикладні програми для запиту і отримання від операційної системи, телекомунікаційного протоколу або програмного інтерфейсу (механізму) пов'язаного з ним обслуговування. Прикладами *API* є *SVID* та *NetBIOS*
ARIN	*American Registry of Internet Numbers* Американський реєстр адрес Інтернет Офіційний сайт – http://www.arin.net/
ARP	*Address Resolution Protocol* Протокол визначення адреси Використовується для отримання *MAC*-адреси, пов'язаної з певною *IP*-адресою
ASN.1	*Abstract Syntax Notation One* Абстрактна синтаксична нотація версії 1 Мова *OSI* для опису абстрактного синтаксису. Мова *ASN.1* визначена в стандартах *CT X.208* и *ISO 8824*. У *SNMP* мова *ASN.1* визначає синтаксис та формат взаємодії між керованими об'єктами та керуючими додатками.
CBC	*Ciphered block chaining* Режим зчеплення шифрованих блоків
CERT	*Computer Emergency Rresponse Team* Група комп'ютерної швидкої допомоги Організація, яка слідкує за загрозами безпеки мережевих комп'ютерів, в тому числі – в *Інтернет*

Cookie	Підтримуваний протоколом *HTTP* текстовий запис розміром до 4 Кбайт з даними про користувача, який повертається веб-сервером при реєстрації користувача та зберігається на його комп'ютері. В цей рядок потрапляє інформація, зібрана сервером про користувач. Початково *cookie* були введені фірмою *Netscape Communications* та описані в *RFC* 2109
Datagram	Дейтаграма Протокольна одиниця обміну (*PDU*) мережевого рівня. Це пакет даних, який містить свою адресу, та передається через мережу з комутацією пакетів незалежно від інших пакетів без розриву логічного з'єднання та квітерування. Містить посилання на попередні пакети, які були адресовані тому ж одержувачу. Дейтаграмний спосіб передачі, який базується на аналізі адреси одержувача, реалізується за допомогою протоколу *IP*
DDoS	*Distributed DOS attacks* Віддалені *DoS* атаки
DES	*Data Encryption Standard* Стандарт шифрування даних Прийнятий в 1980 р. урядом США як стандарт шифрування важливої, але не секретної інформації в державних та комерційних організаціях США. Здійснює шифрування 64-бітових блоків даних за допомогою 64-бітового ключа, в якому значущими є 56 біт (решта використовуються для контролю на парність). Зараз *DES* вважається ненадійним в основному через малу довжину ключа (56 біт) та розмір блока (64 біти). В 2000 р. був офіційно замінений на *AES*
DNS	*Domain Name System* Служба доменних імен 1. Інтернет-служба, розповсюджена по всій земній кулі база даних для ієрархічної системи імен мереж та комп'ютерів, які підключені до *Інтернет*. 2. Протокол прикладного рівня перетворення імені хосту (комп'ютера або іншого мережевого пристрою) в *IP* адресу. Визначений в *RFC* 1034, 1035 та ін. др. Протокол працює поверх протоколу *UDP*, номер порту – 53
DOI	*Domain of Interperetaion* Область інтерпретації Містить відомості, необхідні для встановлення відповідності одних документів іншим. Це, зокрема, ідентифікатори перевірених алгоритмів шифрування й автентифікації, а також деякі параметри, наприклад, тривалості життєвого циклу ключів
DoS	*Denial-of-service* Атаки на відмову в обслуговуванні Зловмисні дії, що виконуються для заборони легальному користувачеві доступу до системи, мережі, додатку або інформації. Атаки *DoS* мають багато форм, вони бувають централізованими (запущеними від однієї системи) або розподіленими (запущеними від декількох систем)
EICAR	*European Institute for Computer Antivirus Research* Європейський інститут комп'ютерних антивірусних досліджень.
ESP	В протоколі *IPSec* *Encapsulating Security Payload* Протокол захищеного корисного вантажу або захищеного вмісту

Ethernet	Базова технологія локальних обчислювальних (комп'ютерних) мереж з комутацією пакетів, що використовує протокол *CSMA/CD* (множинний доступ з контролем несучої та виявленням колізій). Протокол канального рівня дозволяє в кожний момент часу лише один сеанс передачі в логічному сегменті мережі. При появі двох і більше сеансів передачі одночасно, виникає колізія, яка фіксується станцією, що ініціює передачу. Станція аварійно зупиняє процес і очікує закінчення поточного сеансу передачі, а потім знову намагається повторити передачу. Регламентується стандартами *IEEE 802.3* и *ISO 8802.3*
FDDI	*Fiber Distributed Data Interface* Розподілений волоконний інтерфейс даних Стандарт, запропонований *ANSI* для локальних мереж на оптоволокні. Описує високошвидкісні мережі з методом доступу із передачею маркера на основі оптоволокна. Забезпечує зв'язок між мережами різних типів, має обмеження на довжину кільця (не більше 200 км)
flood	Флуд-атака Метод DDoS-атаки на мережі, при якому на комп'ютер-жертву направляється велика кількість запитів, які той врешті не взмозі обробити. Таким чином, комп'ютер-жертва стає недоступною для інших, легальних, користувачів
FTP	*File Transfer Protocol* Протокол передачі файлів Клієнт-серверний протокол прикладного рівня, який дає можливість абоненту обмінюватися двійковими і текстовими файлами з будь-яким комп'ютером мережі що підтримує протокол *FTP* по мережі *TCP/IP*. Визначений в *RFC 959*. Застосовується в *Інтернет* для роботи з *ftp*-серверами. Використовує два паралельних *TCP*-з'єднання: порт 20 для пересилання даних та порт 21 – для керуючого *TCP*- з'єднання
Handshake Protocol	В протоколі *SSL/TLS* Протокол *Квітерування* (Рукостискання) Протокол дозволяє серверові і клієнтові виконати взаємну автентифікацію, а також погодити алгоритми шифрування, обчислення *MAC* і криптографічні ключі, які будуть застосовуватися для захисту даних, що пересилаються протоколом *SSL*. Протокол *Квітерування* повинен використовуватися до початку пересилання даних прикладних програм
HTML	*HyperText Markup Language* Мова розмітки гіпертекстових документів Текстова мова розмітки, призначена для маркування документів, що містять текст, зображення, гіперпосилання тощо. *HTML*-документи лежать в основі *Web*, і відображаються із допомогою веб-браузерів. Мова розмітки розроблялась консорціумом *W3C*, очікується, що *HTML* буде замінена *розширюваною мовою розмітки гіпертексту* (*XHTML*)
IAB	*Internet Architecture Board* (раніше *Internet Activities Board*) Рада з архітектури Інтернет Комітет *IAB* один з керівних органів *Інтернет*. Затверджує нові протоколи, стандарти, проекти розвитку мережі, правила видачі адрес тощо. Працює на суспільних засадах, однак членами *IAB* можуть стати лише особи, що мають відповідну кваліфікацію та авторитет. *IAB* керує комітетами (групами) *IANA, IETF, IRTF*

ICANN	*Internet Corporation for Assigned Names and Numbers* Інтернет корпорація з присвоєння імен та номерів Міжнародна некомерційна організація, створена восени 1998 за участю уряду США для регулювання питань, пов'язаних з доменними іменами, *IP*-адресами та іншими аспектами функціонування *Інтернету*. Офіційний сайт http://www.icann.org/
ICMP	*Internet Control Message Protocol* Міжмережевий протокол керуючих повідомлень Мережевий протокол, що входить в стек протоколів *TCP/IP*. В основному *ICMP* використовується для передачі повідомлень про помилки й інші виняткові ситуації, що виникли при передачі даних. Також на *ICMP* покладають деякі сервісні функції, зокрема на основі цього протоколу заснована дія таких загальновідомих утиліт як ping та traceroute. Протокол *ICMP* не є протоколом орієнтованим на з'єднання (як наприклад *TCP*), тобто при втраті пакета *ICMP* не робитиме ніяких спроб щодо його відновлення. Описаний в *RFC 792* (з доповненнями в *RFC 950*) і є стандартом *Інтернет* (входить в стандарт *STD 5* разом з протоколом *IP*). Хоча формально *ICMP* використовує *IP* (*ICMP* пакети інкапсулюються в *IP* пакети), він є невід'ємною частиною *IP* й обов'язковий при реалізації стека *TCP/IP*. Поточна версія *ICMP* для *IPv4* називається *ICMPv4*. В *IPv6* існує аналогічний протокол *ICMPv6*
ICV	Поле протоколу *IPSec* *Integrity Check Value* Код контролю цілісності
IDS	*Intrusion Detection System* Система виявлення вторгнень Система для виявлення спроб несанкціонованого доступу як ззовні, так і всередині мережі, захисту від атак типу «відмова в обслуговуванні». Використовуючи спеціальні механізми, системи виявлення вторгнень здатні попереджувати шкідливі дії, що дозволяє значно знизити час простою внаслідок атаки і витрати на підтримку роботоздатності мережі
IETF	*Internet Engineering Task Force* Цільова група інженерної підтримки *Інтернет* Відкрите міжнародне співтовариство проектувальників, учених, мережевих операторів і провайдерів, створене *IAB* в 1986 році, яке займається розвитком протоколів і архітектури *Інтернет*. Вся технічна робота здійснюється в робочих групах *IETF*, що займаються конкретною тематикою (наприклад, питаннями маршрутизації, транспорту даних, безпеки тощо). Робота в основному ведеться через поштові списки, але тричі на рік проводяться збори *IETF*. Результати діяльності робочих груп оформляються у вигляді робочих проектів *(Internet drafts)*, які потім використовуються *ISOC* для кодифікування нових стандартів. Основні завдання та функції описані в *RFC 4677*. Офіційний сайт http://www.ietf.org/
IGMP	*Internet Group Management Protocol* Протокол керування групами *Інтернет* Протокол *IGMP* дозволяє окремим користувачам реєструвати на комутаторі або маршрутизаторі, який з'єднує сегменти даної мережі, підключення до певного каналу, щоб отримати розсилання відповідно до специфікації *IP Multicasting*

IP	*Internet protocol* Протокол *Інтернет* Протокол мережевого рівня (частина стеку протоколів *TCP/IP*), який відповідає за передачу та маршрутизацію повідомлень між вузлами *Інтернет*. Найбільш широко розповсюджена реалізація ієрархічної схеми мережевої адресації. Використовуваний в мережі *Інтернет* протокол відповідає за адресацію пакетів, але не відповідає за встановлення з'єднань, не є надійним і дозволяє реалізувати тільки негарантовану доставку даних. Це означає, що протокол для взаємодії не потребує виділеного каналу, як це відбувається під час телефонної розмови, і не існує процедури виклику перед початком передачі даних між мережевими вузлами. Протокол *IP* вибирає найбільш ефективний шлях з числа доступних на основі рішень прийнятих протоколом маршрутизації. Відсутність надійності та негарантована доставка не означає, що система працює погано або ненадійно, а вказує лиш на те, що протокол *IP* не докладає ніяких зусиль, щоб перевірити чи був пакет доставлений за призначенням. Ці функції делеговані протоколам транспортного та вищих рівнів. Транспортний рівень також відповідає за збірку пакетів у повідомлення в потрібній послідовності. Описаний в *RFC 791*. Поточна версія - 4.0, майбутня - 6.0 (*IPv6*)
IPCE	*InterProcess Communication Environment* Дозволяє взаємодіяти процесам, які підтримують протокол *SMTP* в інтерактивному режимі
IPSec	Протокол *IPSec* забезпечує захист обміну даними в локальних мережах, корпоративних і відкритих глобальних мережах і в *Інтернет*. Головною особливістю *IPSec*, яка дозволяє цьому протоколу підтримувати найрізноманітніші програмні додатки, є можливість шифрування і/або автентифікації всього потоку даних на рівні протоколу *IP*
IP-spoofing	Імітація *IP*-адреси Метод атаки на комп'ютерну мережу
ISAKMP	*Internet Security Association and Key Management Protocol* Протокол захищених зв'язків і керування ключами в *Інтернет* *ISAKMP* забезпечує каркас схеми керування ключами в *Інтернет* і підтримку спеціального протоколу і необхідних форматів процедури узгодження атрибутів захисту. *ISAKMP* не змушує використовувати певний конкретний алгоритм обміну ключами, а пропонує набір типів повідомлень, що дозволяють задіяти будь-який подібний алгоритм
IV	В протоколі *IPSec* *Initialization vector* Вектор ініціалізації
Kerberos	Технология *Kerberos* Назва технології автентифікації та шифрування з відкритим ключем, яка створена в середині 1980-х років в Масачусетському технологічному університеті на базі стандарту *DES*. Описана в *RFC 1510*
LAN	*Local Area Network* Локальна комп'ютерна мережа Об'єднання певного числа комп'ютерів (іноді досить великого) на відносно невеликій території. Сучасні локальні мережі будуються на основі топології "зірка" з використанням концентраторів (хабів), комутаторів (свічів) та кабелю *UTP* чи *STP* п'ятої категорії ("вита пара")

MAC	1) *Message Authentication Code* – Код автентичності повідомлення; 2) *MAC-адреса – Media Access Control* – Управління доступом до носія) – це унікальний ідентифікатор, що зіставляється з різними типами устаткування для комп'ютерних мереж. Більшість мережевих протоколів канального рівня використовують один з трьох просторів *MAC*-адрес, керованих *IEEE*: *MAC-48*, *EUI-48* і *EUI-64*. Адреси в кожному з просторів теоретично мають бути глобально унікальними. Не всі протоколи використовують *MAC*-адреси, і не всі протоколи, що використовують *MAC*-адреси, потребують подібної унікальності цих адрес. У широкомовних мережах (таких, як мережі на основі Ethernet) *MAC*-адреса дозволяє унікально ідентифікувати кожен вузол мережі і доставляти дані тільки цьому вузлу. Таким чином, *MAC*-адреси формують основу мереж на канальному рівні, яку використовують протоколи більш високого рівня. Для перетворення *MAC*-адреси в адреси мережевого рівня і назад застосовуються спеціальні протоколи (наприклад, *ARP* і *RARP*)
MD5	*Message Digest* Дайджест повідомлення 128-бітовий алгоритм хешування, розроблений професором Р. Рівестом в 1991 році. Призначений для створення «відбитків» або «дайджестів» повідомлень довільної довжини. Прийшов на зміну *MD4*, що був недосконалий. Описаний в *RFC 1321*
MIB	*Management Information Base* База керуючої інформації Корпоративна база даних, яка містить інформацію про контрольовані та керовані параметри мережевих пристроїв, до яких можливий доступ через протокол керування мережею. База *MIB* розглядається станцією керування як сукупність точок доступу до агента. Об'єкти цієї бази даних стандартні для всіх систем одного класу (наприклад, усі мости підтримують одні й ті ж самі об'єкти керування)
MS Windows	*Microsoft Windows* — узагальнююча назва операційних систем для комп'ютерів, розроблених корпорацією *Microsoft*. http://www.microsoft.com/windows/
MTU	*Maximum Transmission Unit* Максимальний розмір блока даних, що передається Найбільший розмір пакета (фрейму), який може буди переданий по даному фізичному середовищі передачі, наприклад в мережах *Ethernet* ця величина обмежена числом 1518 байт
NFS	*Network File System* Мережева файлова система Стек протоколів для доступу до файлової системи, який оснований на транспортному протоколі *UDP*. *NFS* дозволяє комп'ютерам, які працюють під різними операційними системами) спільно використовувати файли в локальній мережі, позбавляючи необхідності тримати численні копії файлів на локальних дисках окремих комп'ютерів. Розроблена фірмою *SUN Microsystems* в 1984 р. та визначена в *RFC 1094* (версія 2) та *RFC 1813* (версія 3) як відкритий стандарт
Oakley	В протоколі *IPSec* Протокол енергійного обміну ключами Являє собою варіант обміну ключами, який виконується за удосконаленою схемою Діффі-Хеллмана

OSI	*Open Systems Interconnection Reference Model* Модель взаємодії відкритих систем Абстрактна модель для мережевих комунікацій і розробки мережевих протоколів. Подає багаторівневий підхід до мережі. Кожен рівень обслуговує свою частину процесу взаємодії. Завдяки такій структурі спільна робота мережевого обладнання і програмного забезпечення стає набагато простіша й зрозуміліша. Зрозуміло, у наш час основним протоколом є *TCP/IP*, розробка якого не була пов'язана з моделлю *OSI*. Рівні моделі *OSI*: Прикладний рівень (*Application layer*) Рівень подання (*Presentation layer*) Сеансовий рівень (*Session layer*) Транспортний рівень (*Transport layer*) Мережевий рівень (*Network layer*) Канальний рівень (*Data Link layer*) Фізичний рівень (*Physical layer*)
OSPF	*Open Shortest Path First* Першими відкриваються найкоротші маршрути Протокол маршрутизації, за яким на базі топологічної бази даних для передачі пакета першими відкриваються найкоротші маршрути. Стандартний мережевий протокол для маршрутизаторів мережі *Інтернет*
PDU	*Protocol Data Unit* Протокольна одиниця обміну
PGP	*Pretty Good Privacy* Цілком добра приватність Загальнодоступна програма шифрування, яка використовує схему з відкритими ключами. Автор Филип Циммерман (*Philip Zimmermann*)
PKCS#7	В стандарті *X.509* Підписана структура без даних, просто сертифікат чи список сертифікатів, які вже не дійсні
PKI	*Personal Key Infrastructure* Інфраструктура відкритих ключів *PKI*, як і простий сервер-депозитарій, має базу даних для збереження сертифікатів, але, у той же час, надає сервіси і протоколи з керування відкритими ключами. У них входять можливості видання, відкликання (анулювання) цифрових сертифікатів. Головною же особливістю *PKI* є наявність компонентів, відомих як Центр сертифікації (*Certification Authority, CA*) і Центр реєстрації (*Registration Authority, RA*). Центр сертифікації (ЦС) видає цифрові сертифікати і підписує їх своїм закритим ключем. Через важливість своєї ролі, ЦС є головним компонентом інфраструктури *PKI*. Використовуючи відкритий ключ ЦС, будь-який користувач, що бажає перевірити дійсність конкретного сертифіката, звіряє підпис Центра сертифікації і, отже, засвідчується в цілісності інформації, що міститься в сертифікаті, і, що більш важливо, у взаємозв'язку зведень сертифіката і відкритого ключа
PRF	В протоколі *SSL/TLS* *pseudorandom function* Псевдовипадкова функція
proxy	*Proxy server* Сервіс посередництва

	Проксі-сервер – це програма або окремий комп'ютер, який спеціалізується на обробці запитів до мережі і збереженні результатів запитів в своїй локальній кеш-пам'яті. Весь трафік від користувача до серверів в *Інтернет* і назад йде через цей комп'ютер. Проксі-сервер економить час, який абонент витрачає на очікування відповіді з боку сервера (залежно від часу доби популярні сервери можуть бути переобтяжені) і доставку інформації по глобальній мережі. Крім того, іноді проксі-сервер дозволяє отримати інформацію навіть з віддаленого сервера, недоступного зараз, завдяки тому, що ця інформація була раніше кешована
RARP	*Reverse Address Resolution Protocol* Протокол зворотного визначення адреси Протокол із стеку протоколів *TCP/IP*, який використовується для визначення *IP*-адреси вузла локальної мережі, який приєднаний до Інтернет, коли відома лише фізична адреса (*MAC address*). Визначений в *RFC 903*
Record Protocol	В протоколі *SSL/TLS* Протокол запису Протокол забезпечує базовий набір засобів захисту, застосовуваних протоколами більш високих рівнів. Ці засоби, зокрема, може використовувати протокол передачі гіпертекстових файлів (*HTTP*), призначений для забезпечення обміну даними при взаємодії клієнтів і серверів
RFC	*Request for Comments* Запит коментарів Документ із серії пронумерованих інформаційних документів *Інтернету*, які містять технічні специфікації та стандарти, має широке застосування у всесвітній мережі. Назву «*Request for Comments*» ще можна перекласти як «заявка на обговорення» чи «тема для обговорення». Зараз публікацією документів *RFC* займається *IETF* під егідою відкритої організації Товариство *Інтернету* (*ISOC*)
RSA	Схема (алгоритм) асиметричного шифрування з відкритими ключами. Має назву відповідно до прізвищ авторів: *Rivest – Shamir – Adleman*. З деякими змінами ця схема використовується в системах шифрування *PGP encryption*, *PKI*, *private key*, *public key*. Система *RSA* використовується для захисту програмного забезпечення та у схемах цифрового підпису. Через низьку швидкість шифрування, повідомлення звичайно шифрують за допомогою більш продуктивних симетричних алгоритмів з випадковим ключем, а за допомогою *RSA* шифрують лише цей ключ
S/MIME	*Secure Multipurpose Internet Mail Extension* (*Secure MIME*) Безпечний протокол передачі електронної пошти Протокол розроблений компанією *RSA Data Security*. Використовується в багатьох популярних поштових програмах, наприклад у *Netscape Messenger* та *Microsoft Exchange*. Шифрування листів *S/MIME* робить симетричними алгоритмами, а ключ - асиметричними
SET	*Secure Electronic Transaction* Протокол захисту електронних трансакцій Стандарт, який забезпечує безпечний обмін трансакціями при оплаті покупок за допомогою банківських карточок через *Інтернет*. В ньому використовується сертифікаційна схема для перевірки того, що суб'єкт, який здійснює трансакцію дійсно той, за кого себе видає. Запропонований фірмами *Visa International* та *MasterCard*. Офіційний сайт - http://www.setco.org

SHA -1	*Secure Hash Algorithm* Алгоритм автентифікації та перевірки цілісності інформації Алгоритм *SHA* розроблений в США та стандартизований *NIST*. Недолік – 160-бітова функція хешування. Тому *NIST* запропонував три нові функції хешування – SHA-256, *SHA-384* та *SHA-512*
smurf	*Smurf*-атака *DDoS*-атака, при якій зловмисник відправляє *ping*-пакет по широкомовних адресах великої мережі, тоді як за допомогою - спуфінга (*spoofing*) - всі відповіді адресуються до системи-жертви. Якщо проміжна мережа містить багато комп'ютерів, то кількість у відповідь пакетів, направлених до цільової системи, буде такою великою, що призведе до виходу з ладу з'єднання через величезний об'єм даних, що передаються
sniffer	*Сніфер* Слово "*sniffer*" є словом торговельної компанії *Network Associates*, що поширює мережевий аналізатор *Sniffer(r) Network Analyzer*. Сніфером (від англійського *sniff* – винюхувати) називають утиліти для перехоплення мережевого трафіка, адресованого іншому вузлу або в більш загальному випадку – усього доступного трафіка, який проходить або не проходить через даний хост. Більшість сніферів являють собою цілком легальні засоби моніторингу й не потребують установлення додаткового устаткування. Проте їхнє використання в загальному випадку незаконне та потребує відповідних повноважень
sniffing	*Сніфінг* Пасивне прослуховуванням мережі. Для цього сніфер встановлює плату мережевого інтерфейсу в режим прослуховування змішаного трафіка (*promiscuous mode*), тобто мережевий адаптер перехоплюватиме всі пакети, що переміщаються по мережі, а не тільки пакети, адресовані даному адаптеру або системі. Сніфери такого типу добре працюють в мережах з пропускною спроможністю, що розділяється, з мережевими концентраторами – хабами
SNMP	*Simple Network Management Protocol* Простий протокол мережевого керування. Протокол керування мережами зв'язку на основі архітектури *TCP/IP*. *SNMP* – це технологія покликана забезпечити керування та контроль за пристроями і додатками в мережі зв'язку шляхом обміну керуючою інформацією між агентами, що розташовуються на мережевих пристроях, і менеджерами, розташованими на станціях керування. У цей час *SNMP* є базовим протоколом керування мережі *Інтернет*. Визначений початково в 1988 р. у *RFC 1067*, а потім як стандарт в 1990 р. у *RFC 1157*. За замовчуванням використовує порт 161
spoofing	Спуфінг Підміна адреси відправника
SSH	*Secure Shell* Мережевий протокол, що дозволяє проводити віддалене управління комп'ютером і передачу файлів. Схожий за функціональністю з протоколом *Telnet* і *rlogin*, проте використовує алгоритми шифрування інформації, що передається. Криптографічний захист протоколу *SSH* не фіксований, можливий вибір різних алгоритмів шифрування. Клієнти і сервери, що підтримують цей протокол, доступні для різних платформ

SSL	*Secure Sockets Layer* Протокол безпечних з'єднань Протокол для передачі по *Інтернет* зашифрованих, автентифікованих повідомлень, розроблений компанією *Netscape Communications*. Версія *SSL 2.0* прийнята як стандарт *IETF* та застосовується для перевірки повноважень і шифрування даних на транспортному рівні при роботі веб-браузера з веб-сервером. Забезпечує безпеку каналу зв'язку між веб-сервером та веб-браузером. Для доступу до сторінок, які захищені протоколом *SSL*, в *URL* замість звичайного префіксу *http*, як правило, застосовується префікс *https* (порт 443), який вказує на те, що буде використовуватись *SSL*-з'єднання. Остання версія протоколу – *SSL* 3.0 отримала широке розповсюдження. В *IETF* розроблена наступна версія під ім'ям *TLS*
TCP	*Transmission Control Protocol* Протокол керування передачею Основний протокол транспортного та сеансового рівнів в стеку протоколів *TCP/IP*, який забезпечує надійні, орієнтовані на з'єднання, дуплексні потоки. Гарантує доставку переданих пакетів даних в необхідній послідовності, але трафік при цьому може бути досить нерівномірним, оскільки пакети можуть передаватися із затримкою. Протокол *TCP* використовує встановлені логічні з'єднання між клієнтом і сервером та містить механізм контролю перевантаження мережі, забезпечуючи автоматичне зниження швидкості обміну даними. Перша версія визначена в *RFC 793*
TCP/IP	*Transmission Control Protocol / Internet Protocol* Протокол керування передачею / Протокол Інтернет. Фактично *TCP/IP* – це платформонезалежний стек протоколів для комунікації в глобальних та локальних мережах або взаємопов'язаних комплексах мереж (*TCP, IP*). Складається з трьох базових наборів протоколів: *IP* (сервіс нижнього рівня), *TCP* (передача даних) та *UDP*. Надає користувачам два види служб: службу зі встановленням логічного з'єднання (*TCP*) та без встановлення логічного з'єднання (*UDP*). За довгі роки використання в мережах різних країн і організацій стек *TCP/IP* увібрав у себе велику кількість протоколів прикладного рівня. До них належать такі популярні протоколи, як протокол пересилання файлів *FTP*, протокол емуляції терміналу *Ethernet*, поштовий протокол *SMTP*, гіпертекстові сервіси служби *WWW* і багато інших. Сьогодні стек *TCP/IP* є одним з найпоширеніших стеків транспортних протоколів обчислювальних мереж
Telnet	*Telnet* Протокол емуляції термінала 1. Протокол (та відповідні програми) зі стеку протоколів *TCP/IP* для реалізації інтерфейсу мережевого віртуального термінала. Перша специфікація *Telnet* з'явилась як *RFC 318* у 1972 р., потім розвинута в *RFC 854*. 2. Базова мережева послуга, яка дозволяє абоненту *Інтернет* дистанційно підключатися до інших віддалених станцій та працювати з ними зі своєї машини так, наче вона була для них віддаленим терміналом. В наш час, у зв'язку з недоліками безпеки цього протоколу, він майже не використовується – його замінив протокол *SSH*

TLS	*Transport Layer Security* Протокол захисту (безпеки) транспортного рівня Протокол базується на *SSL* 3.0 та повинен прийти йому на зміну. Основна функція протоколу *TLS* полягає в забезпеченні безпеки та цілісності даних між двома взаємодіючими додатками, один з яких є клієнтом, а інший – сервером. Розроблявся на основі специфікації протоколу *SSL 3.0* (*Secure Socket Layer*), опублікованого корпорацією *Netscape*. *TLS 1.0* й *SSL 3.0* несумісні, хоча в *TLS 1.0* передбачений механізм, що дозволяє реалізаціям *TLS* мати сумісність із *SSL 3.0*. Описаний в *RFC 2246*
UDP	*User Datagram Protocol* Протокол дейтаграм користувача Один із протоколів в стеку *TCP/IP*. Від протоколу *TCP* він відрізняється тим, що працює без встановлення з'єднання. *UDP* – це один з найпростіших протоколів транспортного рівня моделі *OSI*, котрий виконує обмін дейтаграмами без підтвердження та гарантії доставки. При використанні протоколу *UDP* обробка помилок і повторна передача даних має виконуватися протоколом більш високого рівня. Але, не зважаючи на всі недоліки, протокол *UDP* є ефективним для серверів, що надсилають невеликі відповіді великій кількості клієнтів. Описаний в *RFC 768*
URL	*Uniform Resource Locator* Уніфікований локатор ресурсів Стандартизована адреса певного ресурсу (такого як документ чи зображення) в *Інтернет*. Придуманий Т. Бернерс-Лі для використання у *WWW*, сучасні форми описуються в *RFC 3986*. Включає в себе назву протоколу доступу (*HTTP*, *FTP*, *telnet*, *gopher* та ін.) і, власне, шлях до ресурсу, формат якого залежить від схеми доступу: *<протокол>://<сервер>[:<порт>][/<шлях>][/<файл>[#<розділ>]]* Для відомих, протоколів, номер порту може не приводитись, тоді використовується стандартний порт (наприклад, порт 80 для *HTTP*)
USM	В протоколі *SNMP* *User-Based Security Model* Модель захисту на базі користувача *USM* забезпечує сервіс автентифікації та таємності в рамках *SNMP*
VACM	В протоколі *SNMP* *View-Based Access Control Model* Модель керування доступом на основі подань Модель *VACM* використовує структуру *MIB*, на основі якої визначається політика керування доступом для даного агента і стає можливим віддалене керування конфігурацією. *VACM* визначає політику керування доступом для даного агента та уможливлює використання віддаленої конфігурації
Wardialing	Розвідка по телефону Метод, використовуваний зловмисниками для виявлення потенційних жертв і визначення систем, що мають модем і які відповідають на вхідні дзвінки
Wardriving	Розвідка на автомобілі Метод атаки, при якому зловмисник роз'їжджає по місту з комп'ютером і адаптером бездротової мережі, виявляючи точки входу бездротових мереж. При цьому використовується пристрій типу *GPS (Global Positioning System* – глобальна система навігації і визначення положення) для запису координат таких точок

Web	*World Wide Web, WWW* 1. Глобальна гіпертекстова система, яка використовує мережу *Інтернет* як транспортний засіб. Мережа серверів, за означенням її засновника Тіма Бернес-Лі (*Tim Bernes-Lee*), – розподілена гетерогенна інформаційна мультимедіа-система колективного використання 2. Сервер, на якому зберігаються *HTML*-документи, пов'язані між собою гіпертекстовими посиланнями
WWW	Див. *Web*
X.25	Серія стандартів *ITU-TSS* (*ITU-Telecommunication Standardization* – сектор стандартизації електрозв'язку Міжнародного союзу електрозв'язку). Визначає протокол, який використовується для пересилання сигналів та даних в мережі з комутацією пакетів. Кожен пакет містить інформацію про відправника та одержувача. Описує цей інтерфейс на трьох рівнях: фізичному, передачі даних та мережевому. Мережі *X.25* отримали свою назву від назви рекомендацій "*X.25*" випущених в 1976 р
X.400	Стандарти *CCITT* (*Consultative Committee for International Telegraph and Telephone* колишня назва Консультативного комітету з міжнародного телеграфного та телефонного зв'язку, МККТТ) для міжнародної електронної пошти (1988 та 1992 рр.). Описують методи електронного обміну текстами, графікою та факсами. Визначають протоколи, які забезпечують надійну передач між агентами користувача та агентами передачі даних
X.509	Стандарт *ITU-T* для інфраструктури відкритого (публічного) ключа (*PKI*) та інфраструктури управління привілеями (*Privilege Management Infrastructure* (*PMI*)). *X.509* визначає стандартні формати для сертифікатів відкритого ключа (*public key certificates*), списку відкликаних сертифікатів (*certificate revocation lists*), атрибутів сертифікатів (*attribute certificates*) та алгоритм підтвердження шляху сертифікації (*certification path validation algorithm*). *X.509* був виданий 3 липня 1988 року
XOR	*Exclusive OR* За виключенням "АБО" Складання за модулем 2. Бінарна логічна операція, результат якої істина тільки тоді, коли значення операндів не співпадають
XSS	*Cross site scripting* Атака, пов'язана з міжсайтовим скриптінгом Досить відома, але як і раніше винятково небезпечна атака. *XSS*-атака звичайно проводиться шляхом конструювання спеціального *URL*, який зловмисник підсуває своїй жертві. *XSS*-атаки дозволяють атакуючому відобразити довільну інформацію й припинити виведення оригінальної веб-сторінки
Алгоритм Діффі-Хелмана	Алгоритм Діффі-Хеллмана *Diffie-Hellman algorithm* Алгоритм Діффі-Хеллмана має дві властивості. 1. Секретні ключі створюються тільки тоді, коли це потрібно. 2. Процедура обміну не потребує ніякої спеціальної підготовки, крім угоди про значення глобальних параметрів. Однак алгоритм Діффі-Хеллмана не позбавлений і недоліків. 1. Алгоритм не пропонує методів, які б дозволили ідентифікувати сторони. 2. Алгоритм вразливий до атак з використанням посередника. 3. Алгоритм потребує інтенсивних обчислень

веб	див. *Web*
Вірус	**Вірус комп'ютерний** (*computer virus*) Комп'ютерна програма, створена для заподіяння шкоди користувачу комп'ютера: знищення і крадіжки даних, пониження роботоздатності комп'ютера тощо
Експлойт	*Exploit* Засіб (програма) для реалізації атаки
З'єднання	*Connection* З'єднанням називається транспорт (у термінах моделі *OSI*), що забезпечує сервіс деякого необхідного типу
Інтернет	*Internet* «Міжмережа» Всесвітня система добровільно об'єднаних комп'ютерних мереж, побудована на використанні протоколу *IP* і маршрутизації пакетів даних. *Інтернет* утворює глобальний інформаційний простір, слугує фізичною основою доступу до веб-сайтів і багатьох систем (протоколів) передачі даних. Сьогодні при вживанні слова «*Інтернет*» найчастіше мається на увазі саме *Web* і доступна через нього інформація, а не сама фізична мережа, що призводить до різноманітних юридичних колізій та правових наслідків. В англійській мові якщо слово «*internet*» написано з малої літери, воно означає просто об'єднання мереж за допомогою маршрутизації пакетів даних. У такому разі не мається на увазі глобальний інформаційний простір
Міжмережевий екран	*Firewall* (Протипожежна стіна) Система (апаратна або програмна) або комбінація систем, яка утворюється з метою захисту границі між двома або більше мережами, попереджаючи від несанкціонованого потрапляння в мережу або виходу з неї пакетів даних. Використовується також для розмежування доступу всередині корпоративної мережі при наявності в ній ділянок з інформацією, яка потребує секретності. Зазвичай функціонує на маршрутизаторах або серверах. Міжмережеві екрани рівня мережі (або пакетний фільтр) досліджують трафік мережі на рівні пакетів мережевого протоколу. Вони можуть, в тому числі, вилучати пакети, щоб дозволити певні типи з'єднань довіреним серверам. Міжмережевий екран рівня додатка досліджує трафік на рівні додатка, наприклад *FTP*, електронної пошти або *Telnet*
Сеанс	*Session* Сеанс *SSL* – це зв'язок між клієнтом і сервером. Сеанси створюються протоколом квітерування *SSL* (*SSL Handshake Protocol*).
Сніфер	Див. *Sniffer*
Спуфінг	Див. *spoofing*
Хеш	Див. *hesh*

Навчальне видання

Дудатьєв Андрій Веніамінович,
Войтович Олеся Петрівна,
Каплун Валентина Аполлінаріївна

ЗАХИСТ КОМП'ЮТЕРНИХ МЕРЕЖ ТЕОРІЯ ТА ПРАКТИКА

Навчальний посібник

Редактор В. Дружиніна
Коректор З. Поліщук
Оригінал-макет підготовлено О. Войтович

Підписано до друку 12.06.2010 р.
Формат 29,7х42 ¼. Папір офсетний.
Гарнітура Times New Roman.
Друк різографічний. Ум. друк. арк. 14.1.
Наклад 100 прим. Зам № 2010-108

Вінницький національний технічний університет,
науково-методичний відділ ВНТУ.
21021, м. Вінниця, Хмельницьке шосе, 95,
ВНТУ, ГНК, к.114.
Тел. (0432) 59-85-32.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.

Віддруковано у Вінницькому національному технічному університеті
в комп'ютерному інформаційно-видавничому центрі.
21021, м. Вінниця, Хмельницьке шосе, 95,
ВНТУ, ГНК, к.114.
Тел. (0432) 59-85-32.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.