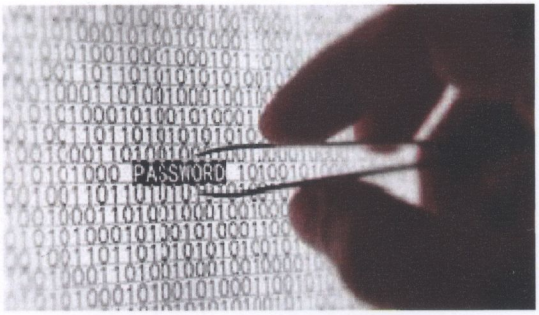
В. А. Каплун
О. В. Дмитришин
Ю. В. Баришев

ЗАХИСТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



Міністерство освіти і науки України
Вінницький національний технічний університет

В. А. Каплун, О. В. Дмитришин, Ю. В. Баришев

ЗАХИСТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Лабораторний практикум

Вінниця
ВНТУ
2017

УДК 681.3.07
ББК 32.973-018.2я73
К20

Рекомендовано до друку Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 12 від 28 травня 2015 р.)

Рецензенти:
В. М. Михалевич, доктор технічних наук, професор
Л. І. Тимченко, доктор технічних наук, професор
С. І. Перевозніков, доктор технічних наук, професор

Каплун, В. А.

К20 Захист програмного забезпечення : лабораторний практикум / Каплун В. А., Дмитришин О. В., Баришев Ю. В. – Вінниця : ВНТУ, 2017. – 75 с.

Лабораторний практикум містить практичні відомості щодо методів захисту програмного забезпечення від несанкціонованого копіювання і використання, від статичного та динамічного дослідження. Детально розглянуто основні засоби програмування для використання цих методів у реалізації задач побудови систем захисту програмного забезпечення.

Посібник призначений для студентів денної і заочної форм навчання із спеціальності 125 «Кібербезпека» (спеціалізації «Безпека інформаційних і комунікаційних систем») при виконанні лабораторних робіт з дисципліни «Захист програмного забезпечення».

УДК 681.3.07
ББК 32.973-018.2я73

ЗМІСТ

Лабораторна робота № 1

ДОСЛІДЖЕННЯ РОБОТИ ПРОГРАМ-МОНІТОРІВ

Мета і задачі

- Ознайомитись на практиці з основними програмними засобами, що використовуються зламниками для попереднього аналізу роботи захищеного програмного забезпечення.
- Дослідити програми моніторингу для виявлення програмами використаних API-функцій і мережевих з'єднань.
- Навчитись обирати програмні засоби для аналізу системи захисту програм.

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Порядок дослідження роботи систем захисту

Для ефективного дослідження (зламу) програмного забезпечення хакерам, перш за все, необхідно зібрати всю необхідну і вичерпну інформацію про роботу цього програмного засобу: про середовище, в якому воно виконується, про глобальні системні змінні та параметри, які він використовує або активно модифікує, про файли і папки, які він використовує для захисту. Крім того, необхідно визначити, чи здійснює дане ПЗ програмні прив'язки та які динамічні бібліотеки використовує.

Наближений алгоритм несанкціонованого дослідження, що його використовує зловмисник для зняття захисту з програми, такий.

1. Перш за все, необхідно запустити програмний засіб, спостерігати за його роботою, за обмеженнями під час роботи та іншими особливостями. Це можуть бути підозрілі рядки, масиви символів, діалогові вікна для введення реєстраційної інформації тощо.
2. Далі необхідно відслідкувати звернення програми до системного реєстру та власних файлів настроювань, оскільки інколи можна лише змінити або видалити певні записи з реєстру або деякий файл для зняття всіх обмежень у роботі програми.
3. Далі потрібно відслідкувати звернення програми до файлів, каталогів і до інших ресурсів файлової системи.
4. Проаналізувати виконувані файли та динамічні бібліотеки на предмет запакування виконуваних файлів, їх зашифрування тощо.
5. Далі можна приступити до дизасемблювання і модифікації програмних модулів, тобто безпосередньо до здійснення зламу.

File Monitors – програми-монітори файлової системи

Даний тип програмних продуктів дозволяє відслідковувати зміни, що відбуваються у файловій системі під час запуску певних програм. У більшості таких програм передбачена система фільтрів для формування протоколів роботи окремих додатків. За допомогою даного типу засобів реалізується *аналіз роботи систем захисту програмного забезпечення (СЗПЗ) з файлами.*

Подібні програми дозволяють з'ясувати, що саме і де змінюють розпізнані на етапах первинного і вторинного аналізу модулі СЗПЗ або визначити модуль, що робить зміни у певному файлі. Ця інформація дозволяє точно локалізувати лічильники кількості запусків ПЗ, приховані файли систем «прив'язки» ПЗ, «ключові файли», файли з інформацією про функції ПЗ, дозволені для використання в рамках даної ліцензії на продукт і т. п., а також модулі і конкретні процедури СЗПЗ, що працюють з цими даними. Важливою є інформація про звернення захищеної програми до файлів, оскільки навіть під час свого виконання програма зчитує команди з власного бінарного файлу до оперативної пам'яті, вже не кажучи про інші приховані файли (файли настроювань, файли ключів). Для вирішення таких задач дослідникам допомагає програма FileMon (рис. 1.1).

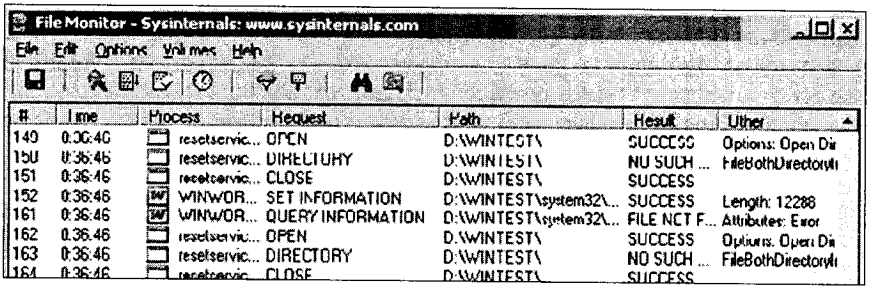


Рисунок 1.1 – Загальний вигляд вікна програми FileMon

У головному вікні програми у таблиці поля містять інформацію про стан об'єкта, про результати виконання операцій з файлами, час виконання певної дії; шлях до файла, з яким відбувається певна дія.

Крім того, повідомлення можна відфільтровувати за певними параметрами та критеріями.

Registry Monitors – програми-монітори системних файлів ОС

Програмні засоби цього типу призначені для відстеження змін, внесених додатками в конфігураційні файли операційних систем (ОС). У розглянутому контексті дані програми дозволяють реалізовувати *аналіз роботи СЗПЗ із системними файлами* (більш специфічно для ОС сімейства Windows).

Розглянуті засоби дозволяють визначати, чи працює система захисту з файлами конфігурації ОС, які зміни вона туди вносить і які дані використовує. У результаті подібного аналізу стає можливим знайти приховані лічильники кількості запусків ПЗ, збережену дату першого установлення ПЗ на ЕОМ користувача, записи з ліцензійними обмеженнями функціональності ПЗ і т. п. Такий аналіз дає результати, подібні до результатів аналізу роботи СЗПЗ з файлами.

Прикладом такої програми може слугувати програма RegMon. Ця програма перехоплює звернення будь-якої іншої програми до системного реєстру, виводячи при цьому повну та докладну інформацію про ключі, до яких здійснювалось звернення (рис. 1.2).

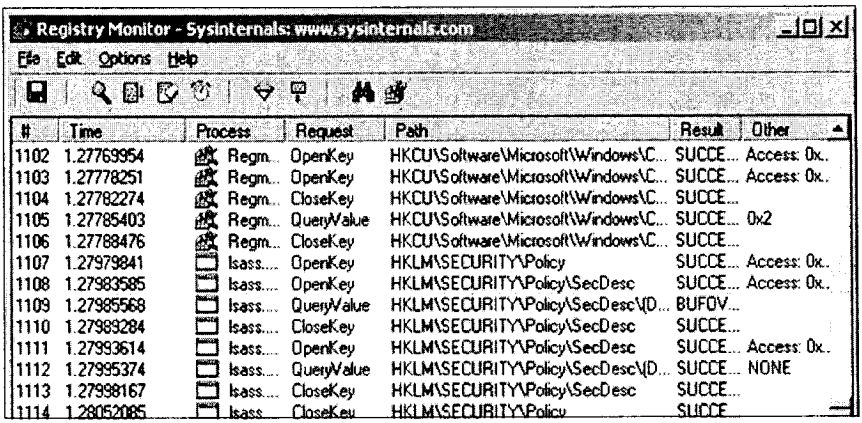


Рисунок 1.2 – Загальний вигляд вікна програми RegMon

Програма також дозволяє фільтрувати повідомлення за певним конкретним процесом.

Знаючи про звернення програми до системного реєстру, можна встановити саме той режим роботи додатка, який необхідний зламнику, або взагалі позбутись всіх обмежень.

API Monitors – програми-монітори викликів підпрограм ОС

Програмне забезпечення цього типу призначено для відстежування виклику системних функцій одним чи декількома додатками з можливістю фільтрації або видалення певних системних функцій або додатків. Застосування таких програм дозволяє проводити *аналіз використання в СЗПЗ системних функцій.*

З огляду на те, що всі дії ПЗ і СЗПЗ, які пов'язані з роботою з файловою системою, роботою з конфігурацією ОС, реалізацією діалогу з користувачем, роботою з мережею та іншим, реалізуються за допомогою викликів функцій ОС. Аналіз використання СЗПЗ системних функцій дозволяє

досить докладно вивчити механізми роботи систем захисту, знайти в них слабкі місця і розробити шляхи обходу впровадженого захисту.

Наприклад, практично всі сучасні системи захисту від копіювання оптичних дисків базуються на досить невеликому наборі системних функцій для роботи з даним видом накопичувачів інформації. Відстежування цих функцій дозволяє знайти та нейтралізувати механізми перевірки типу носія усередині СЗПЗ (рис. 1.3).

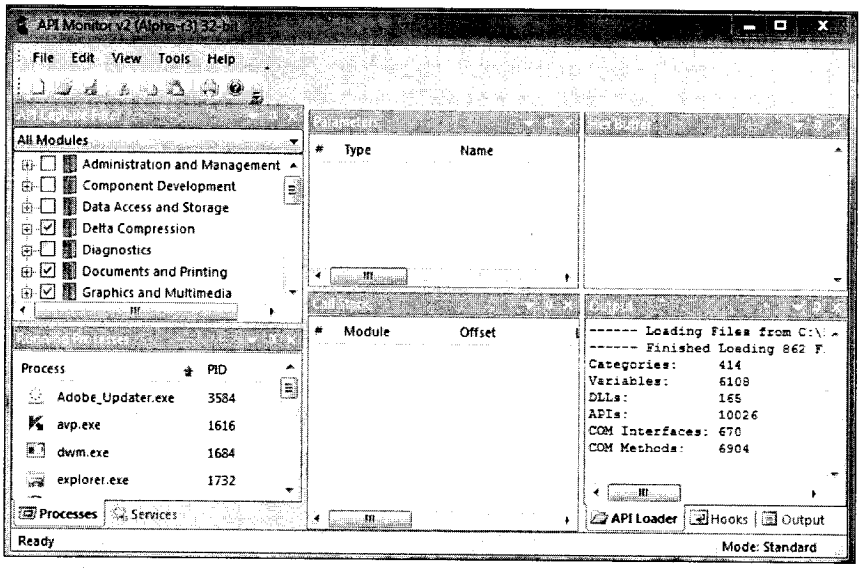


Рисунок 1.3 – Загальний вигляд вікна програми API Monitors

Окрім того, що зазначена програма дає можливість подивитись перелік API-функцій у використовуваній програмі, вона дозволяє подивитись конкретні значення параметрів цих функцій при вході у систему захисту, а, отже, отримати значення паролів, логінів, серійних номерів тощо.

Process/Windows Managers – програми-монітори активних задач, процесів, потоків і вікон

Зазначений тип програмних засобів призначений для стежевання і керування об'єктами ОС (задачами, процесами, потоками, вікнами й ін.). Подібні програми звичайно надають можливості пошуку необхідного об'єкта ОС, переключення на нього керування, зміни його пріоритету, знищення об'єкта, збереження його параметрів (іноді вмісту) на диску. Застосування моніторів задач дає можливість здійснювати *аналіз модульної структури СЗПЗ*. До таких засобів відносять програму Process Explorer (рис. 1.4).

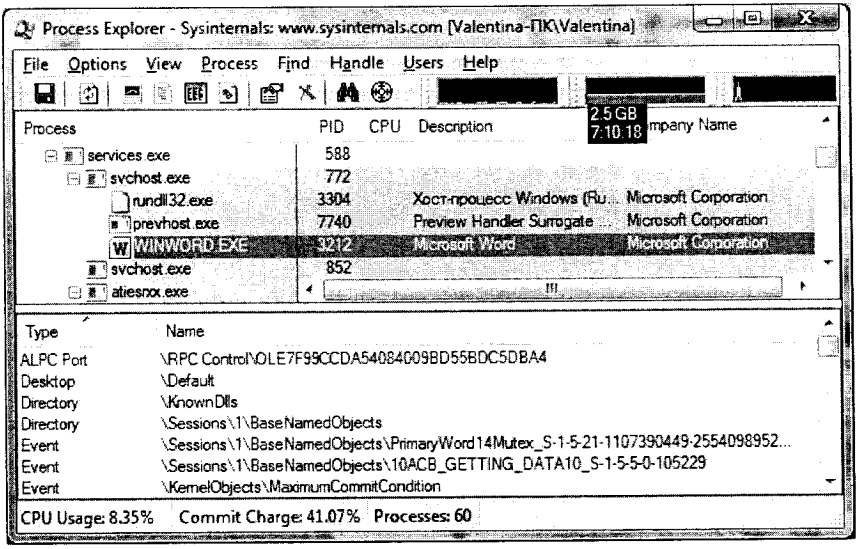


Рисунок 1.4 – Загальний вигляд вікна програми ProcessExplorer

Port Monitors – програми-монітори обміну даними із системними пристроями (портами)

У сучасній архітектурі ОС доступ до всіх системних пристроїв (їх контролерів) здійснюється через порти введення/виведення. Всі сучасні ОС віртуалізують ці порти, організовуючи в такий спосіб спільний доступ декількох додатків до одного й того самого порту (використовуючи механізм черг), а також здійснюючи контроль доступу до портів з метою забезпечення безпеки ОС.

Використання цього типу програмних засобів дозволяє проводити *аналіз взаємодії СЗПЗ із системними пристроями.*

Контролюючи доступ і обмін даними через порти введення/виведення програмної і апаратної частин СЗПЗ, можна аналізувати і переборювати механізми таких типів захистів, як СЗПЗ з електронними ключами, СЗПЗ з ключовими дисками і СЗПЗ «прив'язки» до архітектури комп'ютера користувача.

Прикладом програми для обміну даними із системними пристроями (портами) є програма PortMon (рис. 1.5).

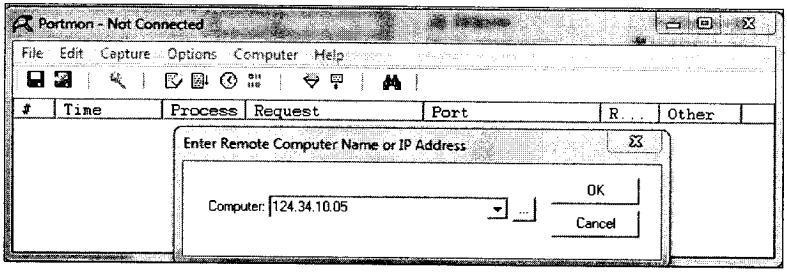


Рисунок 1.5 – Загальний вигляд вікна програми PortMon

Process Monitor – відстеження активності процесів

Програма Process Monitor є вдосконаленим інструментом стеження для Windows, який в режимі реального часу відображає *активність файлової системи, реєстру, а також процесів і потоків* (рис. 1.6). У цій програмі поєднуються можливості двох раніше випущених програм від Sysinternals: FileMon і RegMon, а також величезний ряд поліпшень, охоплюючи розширену та нешкідливу фільтрацію, всеосяжні властивості подій, такі як ID сесій та імена користувачів, достовірну інформацію про процеси, повноцінний стек потоку з вбудованою підтримкою всіх операцій, одночасний запис інформації у файл і багато інших можливостей. Ці можливості роблять програму Process Monitor ключовим інструментом для усунення неполадок та позбавлення від шкідливих програм.

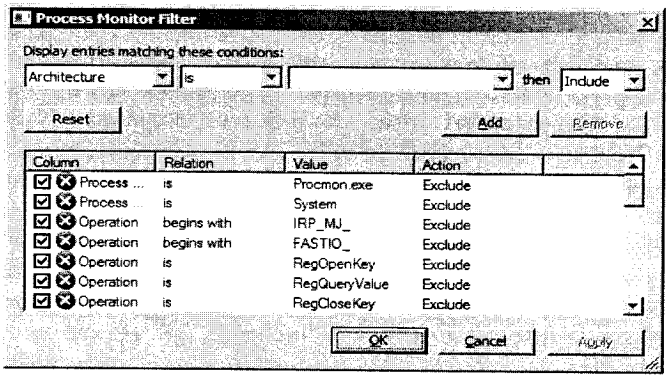


Рисунок 1.6 – Головне вікно програми Process Monitor

Інтерфейс користувача і параметри налаштування програми Process Monitor схожі з інтерфейсом і параметрами програм FileMon та RegMon, але в програмі Process Monitor є ряд істотних поліпшень (рис. 1.7), а саме:

- відстежування запуску та завершення роботи процесів і потоків, охоплюючи інформацію про код завершення;
- відстежування завантаження образів (бібліотек DLL і драйверів пристроїв, що працюють в режимі ядра);
- збір стеків потоків для кожної операції дозволяє в більшості випадків визначити вихідну причину виконання операції;
- достовірний збір інформації про процеси, охоплюючи шлях до образу процесу, командний рядок, а також ID користувача та сесії;
- дерево процесів відображає відносини між всіма процесами, перерахованими у відомостях трасування;
- запис в журнал всіх операцій під час завантаження системи та інші.

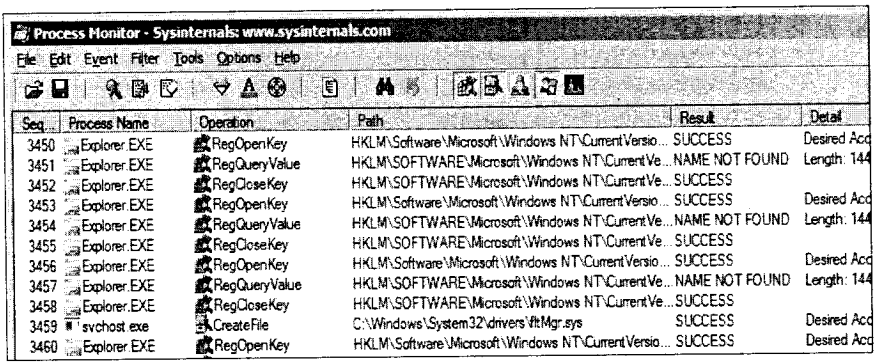


Рисунок 1.7 – Фрагменти інтерфейсу програми Process Monitor

Контрольні запитання

1. Навести загальний порядок здійснення зламу захисту.
2. Охарактеризувати програми-монітори звернень до файлів, їх призначення.
3. Дати характеристику програмам стеження за системним реєстром, їх використанню для аналізу систем захисту.
4. Дати характеристику програмам для моніторингу процесів і вікон.
5. Охарактеризувати програми-монітори API-викликів.
6. Охарактеризувати особливості використання програм сканування портів, програм моніторингу мережевого обміну.
7. Для чого можуть бути використані вказані програми при покращенні роботи операційних систем?
8. Як можуть бути використані програми моніторингу при дослідженні роботи систем захисту програмного забезпечення?

Порядок виконання роботи

1. Ознайомитись з теоретичними відомостями про засоби моніторингу, наведеними в лекціях та в описі даної лабораторної роботи.
2. Ознайомитись з коротким описом програм-моніторів, встановити та запустити на виконання програми, що здійснюють моніторинг різних об'єктів операційної системи (вони знаходяться у папці *Programs*):
   - моніторинг звернень до системного реєстру – *RegMon*;
   - моніторинг звернень до дискової системи комп'ютера – *DiskMon*;
   - моніторинг звернень до файлів та каталогів – *FileMon*;
   - моніторинг звернень до портів – *PortMon*;
   - перегляд інформації про використовувані API-функції – *APIMon*;
   - моніторинг мережного обміну – *Network Traffic Monitors*;
   - моніторинг активності операційної системи – *Process Explorer, Process Monitor.*
3. За допомогою програм-моніторів дослідити роботу таких програм:
   - програми, що містяться у папці Examples;
   - програми, розробленої в результаті виконання курсової роботи з дисципліни «Технології програмування».
4. Проаналізувати отримані результати та оформити звіт з лабораторної роботи за такою формою (бажано у вигляді таблиці за наведеною далі формою):
   - призначення програми, яка є об'єктом дослідження;
   - відомості, які надає кожна з програм моніторингу при слідкуванні за програмою-об'єктом;
   - оцінити кожну програму (за 5-бальною шкалою) відповідно до її функціональності, зручності у використанні, частоті оновлень тощо;
   - зробити висновки щодо ефективності використаного у програмі-об'єкті захисту.

Назва програми	Призначення	Об'єкт дослідження	Результати дослідження	Оцінка	Висновки (позитивні і негативні)
RegMon	Моніторинг звернень до системного реєстру	Programm1	Програма звертається лише до службової інформації і не містить ключової інформації у реєстрі	3	Позитивні: зручний інтерфейс, наявність фільтрів. Недоліки: забагато зайвої інформації, …
…	…	…		…	…

Титульний аркуш до звіту з лабораторної роботи оформити за зразком (додаток А).

Лабораторна робота № 2

ЗАХИСТ ПРОГРАМ ЗА ДОПОМОГОЮ ПРОГРАМ-ПАКУВАЛЬНИКІВ

Мета і задачі

- Дослідити роботу основних програм для захисту виконуваних файлів шляхом пакування.
- Ознайомитись на практиці з програмами для ідентифікації пакування.
- Дослідити програми-розпакувальники, використовувані зламниками для зняття захистів.

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Пакувальники і їх відмінності від архіваторів

Для захисту програмного забезпечення від несанкціонованого дослідження і використання розробники використовують пакувальники. Пакувальники, як правило, зменшують обсяг виконуваних програм, залишаючи при цьому їх повністю функціональними. А от одним із основних кроків, що їх використовують зламники захистів програмного забезпечення, є використання розпакувальників. Проаналізувавши виконувані файли та динамічні бібліотеки на предмет запакування та їх шифрування, зламники можуть застосувати певні програмні засоби для зняття захистів.

Архіватори усім відомі. Приклади таких програм – популярні в Інтернеті WinZip, WinRAR та 7Zip. Архіватор дозволяє запаковувати файли. Проте, в чому полягає запаковування або, як ще говорять, ущільнення?

Спрощено це можна описати так. Використовуючи спеціальні алгоритмічні методи, архіватор знаходить у файлах часто повторювані послідовності і заміняє їх більш короткими кодами. Наприклад, в текстових файлах часто повторюються літери «е», «а» або знак проміжку. Архіватор обчислює число входжень символів у тексті, а потім будує оптимізовану таблицю символів, в якій найбільш використовувані символи мають найкоротші коди, як в азбуці Морзе. Весь текст перекодовується відповідно до нової таблиці, і процес повторюється спочатку. Адже часто зустрічаються не лише окремі літери, але і їх послідовності, наприклад, сполучення «пр», «ст» або «,». Перекодування файлу завершується, коли його розмір після оптимізації перестає зменшуватись.

Зазвичай, архіватори дозволяють ущільнювати не лише текстові файли. Оскільки текст поданий у вигляді набору кодів, архіватор з тим же успіхом ущільнює і нетекстові файли. Наприклад, у файлах програм типу .EXE або .DLL деякі коди команд зустрічаються частіше. Окрім цього, багато компонувачів здійснюють вирівнювання, при якому окремі сегменти

програми доповнюються нулями до тих пір, поки розмір сегмента не стане кратним визначеній величині, наприклад, 8 або 16 байтам. Відповідно, програмні файли містять багато «води», яку можна «вижати» архіватором. В середньому, файли програм ущільнюються приблизно на 40–50%.

Ущільнення досить вигідне для поширення інформації, оскільки дозволяє без втрат розмістити її на носії меншого розміру і значно скоротити час завантаження по мережі. Але архіви, створені звичайними програмами ущільнення, мають один вагомий недолік: файли в них недоступні безпосередньо. Для подальшого використання файли потрібно розпакувати із архіву. Як правило, це виконується тією самою програмою, що використовувалась для запакування. Також існують і саморозпаковувальні архіви, але суті справи це не змінює. Просто для розпакування такого архіву жодних додаткових програм не вимагається. А файли з нього для прямого використання все одно недоступні.

Але існує багато великих програм, якими ми користуємось рідко, але утримувати їх в архіві незручно. Зручніше запускати програму, не розпаковуючи. Давно створений цілий клас програм спеціально для цієї мети. Називають їх пакувальниками. Вони дозволяють упаковувати програму, отримуючи виконуючий файл меншого розміру. Робота пакувальника багато в чому схожа на роботу архіватора. По суті, пакувальник є спеціалізованим архіватором, що створює виконуючі файли програм. Результат його роботи – саморозпаковуваний архів, що містить програму. При запуску програма сама себе розпаковує, а потім починає працювати.

Дії пакувальника такі. Основний код програми (EXE, DLL та ін.) посегментно запаковуть тим або іншим методом (LZW, ZIP та ін.). Потім в початок програми додається процедура її розпакування перед виконанням, і модифікований файл записується на диск. Пакувальники програм для Windows ще витягують із файлу ресурсів основну іконку додатка, щоб файл програми в «Провіднику» виглядав достойно.

Варіантів пакування і розпакування досить багато. Найпростіший варіант: програма при запуску розпаковується повністю. В деяких випадках, спрямованих на боротьбу проти хакерів, частини програми розпаковуються лише по мірі звернення до них (понижаючи швидкодію). Більшість пакувальників працюють по методу «із пам'яті в пам'ять», тобто програма розпаковується в новий сегмент пам'яті і потім з нього запускається.

Отже, ***позитивні риси пакувальників*** такі:

- упакована програма залишається повністю функціональною;
- пакування програми є найпростішим методом захисту її від зламу непрофесійним хакером-крекером;
- крім того, ущільнена програма певним чином захищена від вірусів, оскільки її основний код недоступний для модифікацій вірусом.

Однак, існує ряд ***негативних моментів пакувальників***.

1. Пакування програми збільшує час завантаження: перш ніж починати роботу, програма повинна бути розпакована. Чим більша програма, тим більше часу може витрачатися на її підготовку до виконання.

2. Процедура розпакування програми займає пам'ять. Адже при запуску пам'ять виділяється не програмі, а її розпакувальнику. Після розпакування програма робить запит на нову пам'ять для своєї роботи, розмір якої завжди буде більший, ніж початково виділений даному процесу. Перерозподіл пам'яті і копіювання розпакованої програми може значно знизити швидкодію системи. Це досить актуально для Windows: всі програми, які виконуються у віртуальному адресному просторі, і деякі частини програми можуть бути вивантажені диспетчером пам'яті на диск. А це призводить до зростання кількості звернень до диска і збільшення розміру файлу підкачки.

Отже, пакування програми не так вигідне, як здається з першого погляду. Тим не менше, пакування файлів все таки ускладнює несанкціоноване дослідження програм дизасемблерами.

Пакувальники

UPX Packer

UPX Packer – це поширений пакувальник виконуваних файлів форматів COM, EXE та DLL-бібліотек, який на 50–70% зменшує розмір файлу та час його завантаження. Завдяки цьому програму практично неможливо дизасемблювати, оскільки вона повинна розпаковувати себе під час запуску. Програма має зручний інтерфейс, надає можливість задавати параметри пакування та створити резервні копії файлу (рис. 2.1).

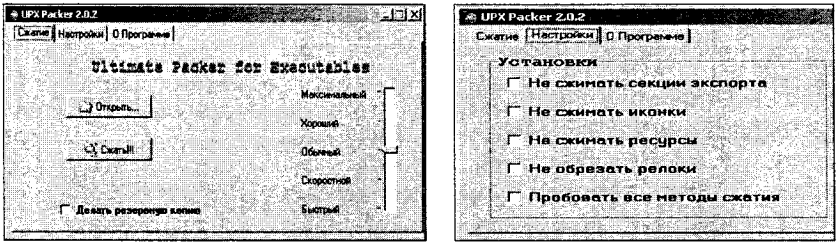


Рисунок 2.1 – Вигляд головного вікна програми UPX

Програма після пакування є абсолютно дієздатною, лише під час запуску вона розпаковується, що займає дещо більше часу на її виконання.

Звичайно, при інших режимах пакування ступінь ущільнення є іншим (при цьому процес пакування проходить значно швидше), а саме: режим «Хороший» – 64,5%; «Звичайний» – 63%; «Швидкісний» – 57,5%; «Швидкий» – 56%.

UPX X-Shell

UPX X-Shell – зручний пакувальник-розпакувальник, призначений для пакування виконуваних файлів. Має дуже зручний інтерфейс (рис. 2.2).

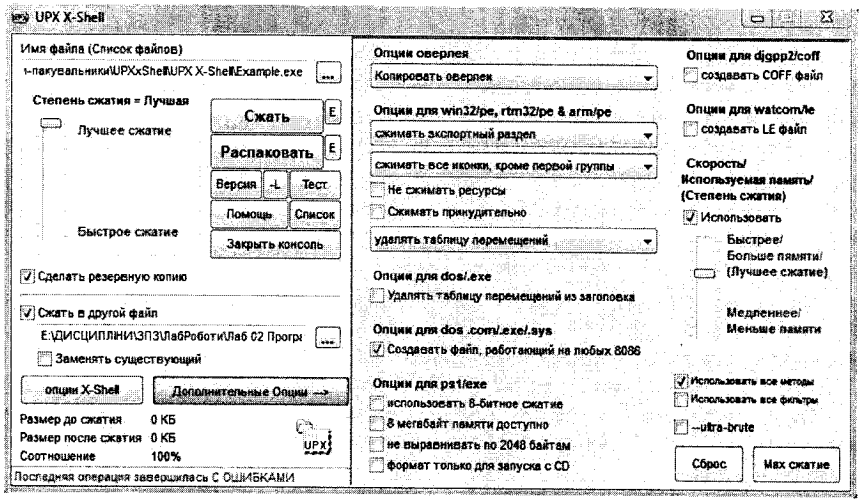


Рисунок 2.2 – Вигляд головного вінка програми UPX X-Shell

Програма дозволяє здійснювати багато налаштувань, вибирати якість та швидкість пакування, використовувати різні методи пакування і т. д., від чого залежить і обсяг упакованого файлу.

ASPack

ASPack – удосконалена програма для ущільнення виконуваних файлів і їх захист від непрофесійного аналізу та реверсивного інженірингу. Дана програма зменшує розміри файлів EXE і DLL-бібліотек Windows до 40–70% (ступінь ущільнення перевищує стандарт ZIP на 10–20%), а також скорочує час завантаження таких додатків у мережах Інтернет. Програми, запаковані ASPack, є автономними і запускаються так само, як і до пакування, без втрати часу і погіршення якостей. Особливості програми:

– шифрування і ущільнення програмного коду, даних і ресурсів (рис. 2.3);
– захист ресурсів і коду від дизасемблерів, декомпіляторів і дамперів;
– цілком прозора, автономна робота з підтримкою довгих імен файлів;
– швидка процедура розпакування порівняно з продуктами конкурентів;
– безпосереднє інтегрування в оболонку Windows, що спрощує схему роботи;
– сумісність з виконуваними файлами, згенерованими Visual C++, Visual Basic, Inprise (Borland) Delphi, C++ Bilder та ін.

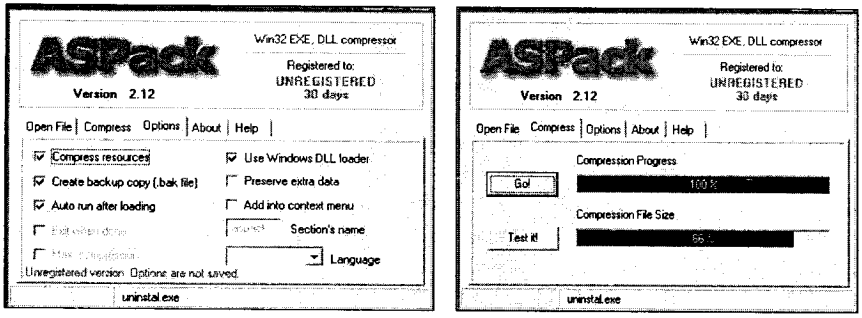
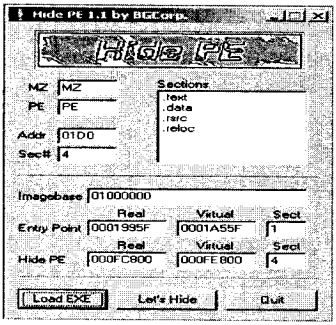


Рисунок 2.3 – Вигляд вікон пакувальника ASPack

Легко переконатись, що після пакування файлу упакована програма продовжує працювати. Для провірки можна натиснути кнопку «Test It!» (або можна зайти в директорію з програмою і запустити її).

Hide PE

Даний пакувальник дещо відрізняється від інших. Сутність захисту полягає у тому, щоб, по-перше, виключити можливість автоматичного розпакування програм і, по-друге, підмінити інформацію про реальний компілятор на іншу, тобто зламник буде вважати, що файл захищено одним пакувальником, тоді як він запакований іншим (рис. 2.4).



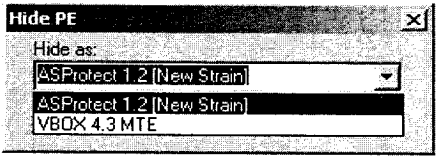


Рисунок 2.4– Вигляд вікон програми HidePE із завантаженим файлом

PE Compact

Дана програма призначена для захисту ПЗ від дослідження і використання і працює з *.EXE та *.DLL файлами (рис. 2.5). Для кожного файлу можна налаштувати власні опції пакування: можливість вибрати ресурси, що підлягають ущільненню, плагіни перехоплювачів API, рівень ущільнення, введення водяного знака тощо.

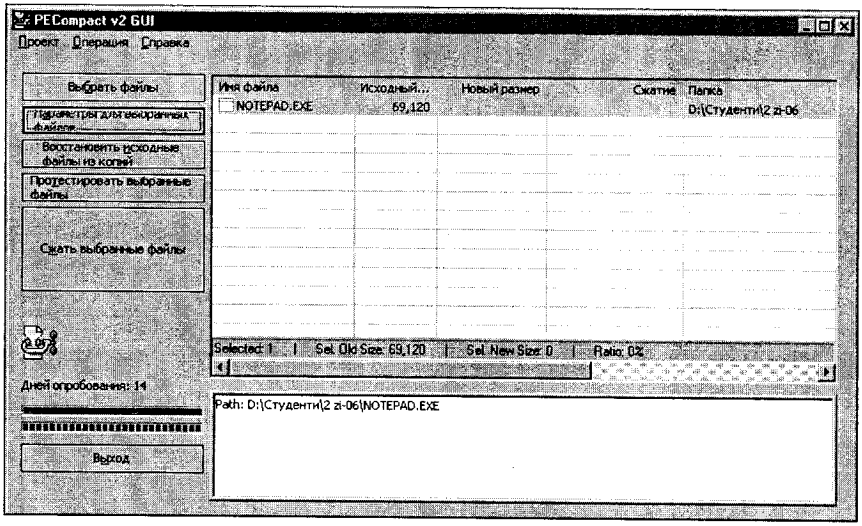


Рисунок 2.5 – Вигляд головного вікна програми PE Compact

Крім наведених, є цілий ряд різноманітних пакувальників, інформацію про які легко знайти у мережі.

Програми-розпакувальники

Якщо існують програми-пакувальники, то повинні існувати і програми-розпакувальники. Адже, як зазначалось раніше, пакування програми захищає її від злому непрофесіоналом. Розпакувальників також є чимало.

Існує два принципово різних методи розпакування запакованих програм:

- метод, орієнтований на конкретний пакувальник;
- розпакування методом трасування.

Розпакувальники першого типу інколи входять до складу пакувальників, наприклад, PKLite. Суть даного методу проста: розпакувальник виконує дії з програмою, протилежні тим, що виконав пакувальник. Перевага такого підходу – отримання в результаті розпакування програми, повністю ідентичної тій, що була раніше. Проте, існує недолік: розпакувальник прив'язаний до конкретного пакувальника або навіть до окремої версії. Якщо файл не розпізнаний як «власний», то розпакувальник його обробляти відмовляється.

Другий метод розпакування, – трасування, – принципово інший. Він був створений хакерами. Трасувальник запускає програму на виконання, а потім входить в режим відлагоджувача і намагається «засікти» той момент, коли розпакування програми закінчиться, і їй буде передане керування. Якщо це вдалось, трасувальник знаходить сегменти програми, виконує їх

компонування і записує на диск. Отриманий в результаті файл не буде байт в байт ідентичний вихідній програмі, точно як методи компонування відрізняються у трасувальника та вихідного компонувальника програми. З іншої сторони, для операційної системи порядок запису сегментів у файлі не має ніякого значення, якщо програма працює правильно. Таким чином, трасування є універсальним способом розпакування програм.

Правда, не завжди вдається «упіймати» момент запуску розпакованої програми. Існують методи маскування, направлені проти трасувальників. Один з них розглядався при огляді пакування програм – розпакування частин програми у міру звертання до них. Якщо під час виконання програма ніколи не розпаковується повністю, то жоден трасувальник не допоможе. Хоча, як і методи захисту, методи зламу постійно удосконалюються.

Ідентифікація параметрів пакування виконуваного файлу

Із зазначеного вище ми знаємо, що для захисту програм розробники використовують програми для пакування виконуваних файлів. Це стає «перешкодою» зловмиснику для здійснення зламу захищеної програми.

Для визначення якою програмою-пакувальником упаковано захищуваний файл, можна скористатись програмою PEId. Дана програма також дозволяє аналізувати не один файл, а й всі файли в обраній папці. Також є можливість рекурсивного аналізу, що охоплює всі підпапки обраної нами папки.

Програма має вбудований менеджер задач, що дозволяє переглянути, які динамічні бібліотеки використовують програми в даний час (рис. 2.6).

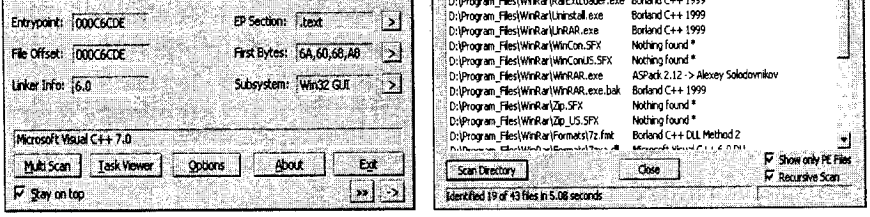


Рисунок 2.6 – Вигляд головного вікна програми PEId і вікна мультисканера

Крім того, програма дозволяє переглянути й іншу інформацію про виконуваний упакований файл, а саме:

- заголовок виконуваного файлу (рис. 2.7);
- секції виконуваного файлу (рис. 2.8).

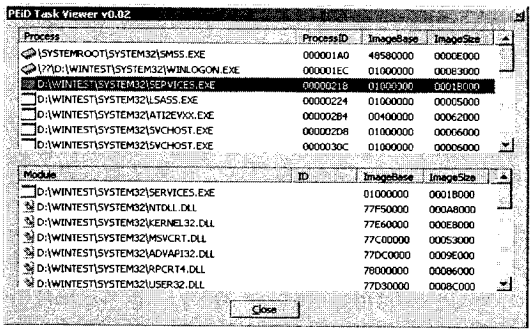


Рисунок 2.7 – Вигляд вікна менеджера задач та списку модулів PEId

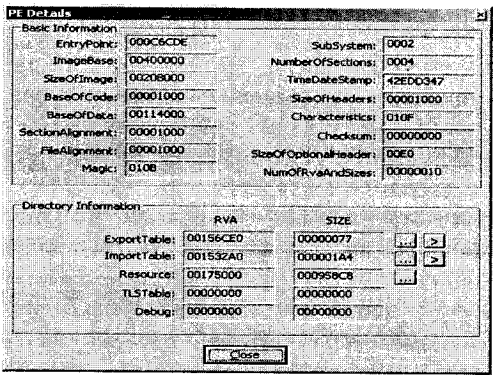


Рисунок 2.8 – Вигляд інформації про заголовок виконуваного файлу

У Інтернет-мережі можна знайти багато цікавої інформації і про пакувальники, і про розпакувальники, і про програми ідентифікації пакування, а також багато конкретних прикладів цих програм.

Контрольні запитання

1. В чому різниця між архіваторами та пакувальниками?
2. Який принцип дії програм-пакувальників?
3. Які позитивні і негативні риси пакувальників?
4. Назвіть декілька програм для пакування виконуваних файлів.
5. Наведіть приклади програм для ідентифікації пакування файлів. Яке їх призначення?
6. Яке програмне забезпечення виконує злам програмних продуктів, захищених пакувальниками?
7. Назвіть основні методи, що їх використовують розпакувальники.

Порядок виконання роботи

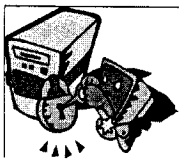
1. Ознайомитись з теоретичними відомостями про програми пакування, розпакування виконуваних файлів.
2. Підготувати для роботи декілька програм (результати виконання лабораторних і курсових робіт I-II курсів):
   - виконуваний файл консольної програми *малого обсягу*;
   - виконуваний файл консольної програми *великого обсягу*;
   - виконуваний файл API-додатка під Windows;
   - виконуваний файл MFC-додатка під Windows.
3. Підготувати не менше п'яти програм-пакувальників (завантажити з Інтернету сучасні версії програм).
4. Підготувати дві програми для ідентифікації пакування програм.
5. Для кожної з програм-об'єктів виконати такі дії:
   - запустити на виконання програму, попередньо зафіксувавши її обсяг;
   - запакувати файли по черзі кожним з пакувальників, попередньо ознайомившись з їх можливостями і особливостями. При цьому зафіксувати обсяг і час виконання запакованих програм. Зберігати всі варіанти запакованих програм (для проведення подальших досліджень);
   - дослідити захищену програму на предмет можливості розпакування програмами для ідентифікації пакування після кожного пакування;
   - розпакувати запаковану програму, використовуючи відповідні розпакувальники, і дати характеристику якості розпакування. Порівняти отриманий в результаті розпакування файл з початковим.
6. Оформити *звіт*, в якому вказати (результати звести у таблицю за наведеною далі формою):

Назва програми-об'єкта, її обсяг	Пакувальник	Можливості пакувальника	Результати пакування		Працює (так/ні)	Програма ідентифікації пакування	Результат ідентифікації	Розпакувальник	Результат розпакування	
			Обсяг	% збільш. зменш.					Обсяг	Працює (так/ні)

7. Сформувати висновки щодо методів та принципів роботи програм пакування і розпакування файлів та програм-ідентифікаторів пакування. Оцінити кожну з програм-пакувальників і розпакувальників за п'ятибальною шкалою, пояснити і обґрунтувати свою оцінку.

Лабораторна робота № 3

ЗАХИСТ ПРОГРАМ ВІД КОПІЮВАННЯ ШЛЯХОМ ПРИВ'ЯЗКИ ДО ФІЗИЧНИХ ПАРАМЕТРІВ



Мета і задачі

- Ознайомитись з основними поняттями ОС, необхідними для створення захисту програмного забезпечення від несанкціонованого копіювання.
- Ознайомитись з методами отримання параметрів комп'ютерної системи.
- Засвоїти основны методи захисту програмного забезпечення за допомогою прив'язки до носіїв для запобігання копіюванню.
- Дослідити існуючі програми захисту від копіювання.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Способи захисту від несанкціонованого копіювання

Захист програмного продукту від несанкціонованого копіювання – актуальна задача у зв'язку зі збереженням комерційних і авторських прав фірм і розробників. Термін «захист від копіювання» досить умовний, оскільки практично завжди можна переписати інформацію, що знаходиться на будь-якому носії інформації. Інша справа, що після цього програма може не виконуватися. Таким чином, без санкції розробника чи фірми-розповсюджувача неможливо одержати дієздатний програмний продукт. Тобто, «захист від копіювання» – це створення засобів, що надають можливість «захисту від несанкціонованого виконання».

Системи захисту від несанкціонованого копіювання можна розділити на такі групи:

- прив'язка до дистрибутивного носія (диска, CD- або DVD-диска);
- прив'язка до комп'ютера (до архітектури, до технічних характеритик або до штатного програмного забезпечення);
- прив'язка до ключа (ЕК або SecretKey);
- опитування довідників;
- обмеження на використання ПЗ.

Перші три групи методів захисту від копіювання пов'язані з роботою з дисками на фізичному рівні, а для цього необхідно добре знати будову дисків та можливості керування ними.

Однією з розповсюджених технологій захисту від НСК є створення особливо обумовлених носіїв. Їх особливість полягає в тому, що на носії створюється спеціально організована мітка, яка використовується як ознака її дистрибутивності. Функцію контролю мітки виконує спеціальна частина програми, що захищається. Після копіювання захищеного носія

засобами ОС буде скопійована вся інформація, за винятком мітки. Під час виконання програми її контролююча частина (КЧП) встановить, що диск не дистрибутивний, і припинить виконання програми. Тим самим програма ніби «прив'язується» до свого носія.

Методи отримання повної інформації про комп'ютерну систему

Існує декілька методів для отримання необхідної інформації про апаратну частину системи, яка нас цікавить.

По-перше, це опосередковане звертання до BIOS через переривання BIOS та DOS.

По-друге, можна звернутись безпосередньо до розділу пам'яті BIOS (копія таблиці параметрів в пам'яті) за його абсолютною адресою.

Можна використати бібліотеки функцій мови програмування високого рівня або самої операційної системи Windows (табл. 3.1).

Таблиця 3.1 – Способи отримання інформації про комп'ютер

Опис методу	Переваги	Недоліки
Опосередковане звертання до BIOS, через переривання	Можливість отримання повного спектра необхідної інформації Висока швидкодія Зручність у використанні	Необхідність створення коду програми на мові низького рівня Існуючі недоліки переривань
Отримання інформації з пам'яті BIOS за його абсолютною адресою	Найбільша швидкодія	В пам'яті BIOS знаходяться лише деякі основні дані Незручність у використанні
Використання бібліотеки функцій мови програмування високого рівня	Дуже зручно у використанні	Найменша швидкодія Функції високого рівня мають ряд власних дефектів
Використання бібліотеки функцій операційної системи Windows	Дуже зручно у використанні. Висока швидкодія. Висока точність	Необхідна наявність документації для цих функцій

Оскільки для захищуваного програмного забезпечення одним з критичних ресурсів є час, то оптимальним буде використання комбінації двох перших найшвидших методів.

Серед програмних бібліотек є бібліотека, яка містить заголовний файл *Sysinfo.h*, у якому описано клас *Sysinfo*, що містить величезну кількість

полів і цікавих функцій. За допомогою цих функцій можна отримати багато різноманітної інформації про комп'ютерну систему, щоб надалі здійснити прив'язку до них як до унікальних параметрів (табл. 3.2).

Таблиця 3.2 – Інформація, використовувана для «прив'язок»

Розділ	Інформація	
Тип комп'ютера	Ідентифікатор машини Додатковий код моделі	
Процесор	Тип і модель процесора Тактова частота процесора Сімейство процесора	Тип співпроцесора Фірма-виготовлювач процесора Сигнатура процесора і т. д.
BIOS	Модель BIOS. Дата BIOS	
Відеоадаптер	Тип, назва моделі відеоадаптера	Обсяг відеопам'яті
Диски	Кількість FDD HDD дисків Виготовлювач HDD Модель HDD Серійний номер HDD	Контрольний номер HDD Кількість голівок циліндрів HDD Розмітка HDD
Пам'ять	Розмір основної пам'яті Розмір вільної основної пам'яті Адреса керуючої програми драйвера XMS	Розмір пам'яті XMS Розмір вільної пам'яті XMS Розмір пам'яті EMS Розмір вільної пам'яті EMS
CD ROM	Кількість пристроїв. Номери пристроїв CD-ROM (букви дисків) Версія драйвера MSCDEX	
Інше	Кількість портів Наявність ігрового порту і підключеного пристрою Тип звукової карти	Тип клавіатури / кількість клавіш Тип миші / кількість клавіш Версія драйвера миші Тип операційної системи

Контрольні запитання

1. Навести характеристику сучасних технологій захисту програмного забезпечення від копіювання.
2. Охарактеризувати рівні роботи з дисковою системою комп'ютерів.
3. Навести перелік методів захисту від НСК шляхом прив'язки до дистрибутивного носія.
4. Що означає поняття нестандартного форматування і яким чином воно використовується для захисту від НСК? Навести приклади.
5. Що означає метод, який базується на опитуванні довідника? Що таке електронні ключі, які є типи елекронних ключів?
6. Що таке ключі ідентифікації, який принцип захисту вони використовують і чим відрізняються від електронних ключів?
7. Знайти в Інтернеті програмні продукти, використовувані для захисту від копіювання. Навести їх можливості, способи захисту, використовувані методи.

Порядок виконання лабораторної роботи

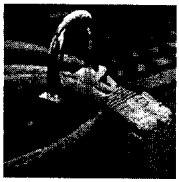
1. Запустити програму *SysInfo.exe.* Проаналізувати основні параметри вашого комп'ютера. Ознайомитись з вихідним кодом програми SysInfo. Дослідити можливості використання класу SysInfo.
   У звіт: характеристики власного комп'ютера; програмні засоби для отримання інформації; висновки і пропозиції щодо можливості використання отриманої інформації при захисті.
2. Запустити програму *Disks* з папки Disks. Проаналізувати отриману інформацію і порівняти характеристики розділів жорсткого диска, flesh-носіїв, CD-дисків і інших носіїв інформації. Дослідити функції, які зостосовуються для отримання інформації про дискову систему комп'ютера.
   У звіт: характеристики переглянутих дисків і їх порівняння (таблиця); програмні засоби для отримання інформації про дискову систему.
3. Дослідити роботу і код програм *Crack05* і *Crack09.* Дослідити їх можливості, спосіб захисту.
   У звіт: охарактеризувати спосіб захисту, навести програмні засоби для роботи з реєстром.
4. Звіт з виконання цієї частини лабораторної роботи можна оформити, звівши всю досліджену інформацію у таблицю за такою формою.

Функція	Параметри і їх значення	Призначення функції

5. Встановити на комп'ютері програму *Orien.* Дослідити її інтерфейс та можливості захисту за її допомогою. Взяти будь-яку виконувану програму (можна використати програму, розроблену під час курсового проектування) і захистити її різними способами за допомогою програми Orien. Впевнитись в тому, що програму справді захищено від копіювання, запустивши її на іншому комп'ютері.
   У звіт: призначення, можливості і варіанти захисту; назва програми-об'єкта, призначення, результати виконання без захисту; вид захисту (використати чотири різних методи захисту), розмір програми після захисту, результати виконання на «рідному» комп'ютері, результати виконання на «чужому» комп'ютері.
6. Знайти в Інтернеті і навести відомості про інші програми, що застосовуються для захисту програм від несанкціонованого використання.
   У звіт: назва програми, виробник, вартість, призначення, режими роботи, методи захисту.

Лабораторна робота № 4

РОЗРОБКА ЗАХИСТУ ПРОГРАМ ШЛЯХОМ ПРИВ'ЯЗКИ ДО ПАРАМЕТРІВ КОМП'ЮТЕРА І СИСТЕМНОГО РЕЄСТРУ



Мета і задачі

- Засвоїти методи прив'язки програмного забезпечення до характеристик комп'ютера з метою захисту його від НСК.
- Дослідити програмні засоби для отримання параметрів комп'ютерної системи і засвоїти їх використання у власних програмах захисту.
- Засвоїти програмні засоби для роботи з системним реєстром і навчитись використовувати їх у власних програмах захисту.

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Основні методи прив'язки до комп'ютера

Одним з методів захисту програм від несанкціонованого копіювання є використання деяких заздалегідь визначених характеристик допущеного для роботи комп'ютера. Це є досить ефективний шлях, оскільки перевірка комп'ютера проста і нетривала, може часто повторюватися за ходом роботи, не знижуючи загальної швидкодії.

Розрізняють такі методи прив'язки до комп'ютера:

– прив'язка до вінчестера (до місця розташування певної програми на диску, до серійного номера і т. д.);
– прив'язка до BIOS (дата створення, вміст CMOS пам'яті тощо);
– прив'язка до архітектури, набору ПЗ, до конфігурації (тип мікропроцесора і розрядність шини даних; наявність і тип контролерів для зовнішніх пристроїв і самих пристроїв; інші особливості);
– вимірювання продуктивності апаратури; частіше за все вимірюється швидкодія процесора, пам'яті, контролерів і т. д.; швидкість обертання двигунів дисководів; швидкість реакції на зовнішній вплив.

Окрім цього, захист від копіювання можна забезпечити шляхом використання електронних ключів. Електронними ключами, в основному, захищають так званий «діловий» софт: бухгалтерські і складські програми, правові і корпоративні системи, будівельні кошториси, САПР, електронні довідники, аналітичний софт, екологічні і медичні програми і т. п.

Електронний ключ запобігає незаконному використанню (експлуатації) програми. Часто говорять, що ключ захищає від копіювання, але це не зовсім правильно. Захищену програму можна скопіювати, тільки копія без ключа працювати не буде. Таким чином, копіювання просто не має сенсу.

Використання системного реєстру при захисті програм

Реєстр часто використовують при захисті програмного забезпечення від несанкціонованого використання, а саме у таких випадках:

– приховування параметрів прив'язки (складових архітектури, характеристик обладнання і т. д.) у системному реєстрі;

– прив'язки до часу використання програми;

– прив'язки до кількості запусків програми.

Коротко розглянемо основні функції роботи з реєстром. Вони описані у заголовному файлі <winreg.h>.

RegCreateKeyEx( )– створення ключів

Перед створенням нового ключа необхідно продумати, в яку гілку дерева необхідно включити новий ключ. Окрім цього, ключ, хендл якого вказаний в першому аргументі, повинен бути відкритий з атрибутом KEY_CREATE_SUB_KEY.

```
RegCreateKeyEx (
HKEY hKey,          // хендл раніше відкритого ключа
LPCSTR lpSubKey,    // покажчик на рядок з іменем створюваного
       // ключа, який є підлеглим ключа з першого аргументу
DWORD Reserved,     // зарезервований і повинен бути рівним 0.
LPSTR IpClass,      // покажчик на рядок - клас ключа
DWORD dwOptions,    // опції створюваного ключа
REGSAM samDesired,       // маска доступу до ключа
LPSECURITY_ATTRIBUTES IpSecurAttributes, // атрибути безпеки
PHKEY phkResult,              // хендл створеного ключа.
LPDWORD lpdwDisposition);     // куди буде записана інформація
                              // про зміни з ключем.
```

dwOptions – визначає опції створюваного ключа, може приймати одне із значень – REG_OPTION_VOLATILE або REG_OPTION_NON_VOLATILE. Друге значення вказує, що при перезавантаженні системи значення цього ключа зберігається, тобто інформація зберігається у файлі, а не в пам'яті.

samDesired – визначає маску доступу до ключа. Цей параметр є бітовою шкалою і може бути комбінацією таких прапорів:

KEY_QUERY_VALUE – має право запрошувати дані підключів;
KEY_SET_VALUE – має право встановлювати дані підключів;
KEY_CREATE_SUB_KEY – має право створювати подключі;
KEY_ENUMERATE_SUB_KEY – має право перебирати ключі;
KEY_NOTIFY – має право змінювати нотифікацію;
KEY_CREATE_LINK – має право створювати символічний зв'язок;
KEY_READ – (STANDART_RIGHTS_READ | KEY_QUERY_VALUE | KEY_ENUMERATE_ SUB_KEY | KEY_NOTIFY) & (~SYNCRONIZE);
KEY_WRITE – (STANDART_RIGHTS_WRITE | KEY_QUERY_VALUE | KEY_CREATE_ SUB_KEY) & (~SYNCRONIZE);
KEY_EXECUTE – KEY_READ & (~SYNCRONIZE);
KEY_ALL_ACCESS – (STANDART_RIGHTS_READ | KEY_QUERY_VALUE | KEY_SET_VALUE | KEY_CREATE_SUB_KEY | KEY_ENUMERATE_SUB_KEY| KEY_NOTIFY | KEY_CREATE_LINK) &(~SYNCRONIZE));

lpSecurityAttributes – покажчик на структуру SECURITY_ATTRIBUTES, яка визначає атрибути безпеки даного ключа.

lpdwDisposition – вказує місце, куди буде записана інформація про те, що відбулося з ключем. Якщо за допомогою цієї функції здійснюється спроба створити ключ, який вже існує, то ключ не створюється, а просто відкривається. Тому необхідно знати, що відбулося при створенні ключа. Якщо ключ був створений, то в lpdwDisposition, записується значення REG_CREATED_NEW_KEY. У випадку, якщо ключ існував і був відкритий, записане значення дорівнюватиме REG_OPENED_EXISTING_KEY.

Функція повертає значення ERRORSUCCESS в тому випадку, якщо ключ створений або відкритий вдало. Будь-яке інше значення є свідченням того, що при створенні або відкритті ключа сталася помилка.

RegOpenKeyEx( ) – відкриття ключів

Вважаємо, що ключ ми створили. Для того, щоб не створити, а відкрити існуючий ключ, потрібно викликати функцію:

```
RegOpenKeyEx (HKEY hKey, LPCSTR lpSubKey,
      DWORD ulOptions,REGSAM samDcsired, PHKEY phkResult);
```

Аргументи цієї функції мають значення, подібні аргументам попередньої функції. Полю Reserved функції RegCreateKeyEx() відповідає поле ulOptions .

RegCloseKey(), RegFlushKey() – закриття ключів і збереження змін

Закривається ключ за допомогою функції ***RegCloseKey()***, опис якої приведений нижче, можна зустріти у файлі <winreg.h>:

```
RegCloseKey (HKEY hKey).
```

Єдиним аргументом цієї функції є хендл ключа, що закривається. Але при виконанні цієї функції можуть виникнути деякі проблеми, непомітні з першого погляду. Справа у тому, що дані з реєстру на час роботи з ними переписуються в кеш і записуються назад на диск при виконанні функції *RegFlushKey()*, опис якої має такий вигляд:

```
RegFlushKey (HKEY hKey).
```

Іншими словами, якщо ми хочемо, щоб дані, які ми змінили під час роботи програми, були втрачені, перед закриттям ключа потрібно скидати на диск. З іншого боку, у програміста може з'явитися спокуса скидати дані на диск досить часто. Оскільки RegFlushKey() використовує величезну кількість системних ресурсів, то цю функцію потрібно викликати тільки у випадку, коли дійсно в цьому є потреба.

Додавання даних до ключів і видалення даних з ключів

Після того, як ключ створений, виникає необхідність додати до ключа деякі дані, які використовуватимуться програмою. Для цього потрібно викликати функцію ***RegSetValueEx()***:

```
RegSetValueEx (
  HKEY hKey, // хендл ключа, до якого додаються дані
LPCSTR lpValueName, // рядок з даними, що додаються
DWORD Reserved,      // зарезервовано
DWORD dwType,          // тип інформації, який буде збережений
CONST BYTE* lpData, // покажчик  на дані, які будуть збережені
DWORD cbData);         // розмір даних, на які вказує 5-й аргумент
```

Четвертий аргумент визначає тип інформації, який буде збережений як дані. Цей параметр може приймати одне із значень:

REG_NONE (0) – тип даних не встановлюється;
REG_SZ (1) – рядок, який закінчується нулем;
REG_EXPEND_SZ (2) – рядок з посиланнями на змінні оточення (типу %PATH%);
REG_BINARY (3) – бінарні дані;
REG_DWORD (4) – подвійне слово;
REG_DWORD_LITTLE_ENDIAN (4) – як REG_DWORD;
REG_DWORD_BIG_ENDIAN (5) – як REG_DWORD, але значущим є молодший біт;
REG_LINK (6) – символічний зв'язок;
REG_MULTI_SZ (7) – масив з декількох рядків;
REG_RESOURSE_LIST (8) – список драйверів пристроїв;
REG_FULL_RESOURCE_DESCRIPTOR (9) – список ресурсів (апаратура);
REG_RESOURCE_REQUIREMENTS_LIST (10).

До речі, одержати повну інформацію про підключі можна за допомогою функції ***RegQueryInfoKey().***

```
RegQueryInfoKey (HKEY hKey, LPTSTR lpClass,
    LPDWORD lpcClass,LPDWORD lpReserved,LPDWORD lpcSubKeys,
    LPDWORD lpcMaxSubKeyLen, LPDWORD lpcMaxClassLen,
    LPDWORD lpcValues,   LPDWORD lpcMaxValueNameLen,
    LPDWORD lpcMaxValueLen,LPDWORD lpcbSecurityDescriptor,
    PFILETIME lpftLastWriteTime.
```

А видалити дані можна за допомогою функції ***RegDeleteValue()***:

```
RegDeleteValue  (HKEY hKey, LPCSTR IpValueName ).
```

Аргументи цієї функції: хендл ключа і покажчик на рядок з іменем даних.

Вибірка даних з реєстру

Якщо прикладній програмі потрібно здійснити вибірку даних з реєстру, то спершу програма повинна визначити, з якої гілки реєстру їй потрібно вибрати дані. Природно, що ніяких функцій для цього немає. При написанні програми програміст повинен сам поклопотатися про це. Програма повинна перебирати всі ключі в цій гілці, поки не знайде потрібний ключ. Для цього додаток може скористатися функцією ***RegEnumKeyEx()***.

Припустимо, що за допомогою способу, описаного вище, ми перебрали підключі і знайшли потрібний нам підключ. Тепер в цьому підключі нам необхідно знайти потрібні дані. Спосіб пошуку такий самий, як і в попередній функції. Для пошуку необхідно перебрати всі дані, пов'язані з підключем. Щоб здійснити цей перебір, звичайно використовують функцію ***RegEnumValue()***.

Отримання параметрів комп'ютерної системи

Функції для отримання параметрів логічних дисків

Наведемо лише прототипи функцій для отримання характеристик комп'ютерної системи. Детальний опис цих функцій можна знайти у відповідній літературі.

GetLogicalDrives() – перелік логічних дисків;
GetLigicalDriveStrings() – список кореневих директорій дисків;
GetDriveType() – перелік типів дисків;
GetVolumeInformation() – детальна інформація про диски;
GetDiskFreeSpace() – інформація про розбиття файлу на сектори і кластери, про загальний і вільний простір на диску);
DeviceControl() – робота з диском на низькому рівні, оперування інформацією з драйверів пристроїв).

Функції для отримання інформації про файли на комп'ютері

SetCurrentDirectory(), *GetCurrentDirectory()* – встановлення і отримання поточної директорії;
CreateDirectory() – створення директорії;
RemoveDirectory() – видалення директорії;
GetSystemDirectory() – отримання системної директорії;
GetWindowDirectory() – отримання директорії Windows;
GetFileInformationByHandle() – узагальнена функція, яка може використовуватися для отримання всієї наведеної вище інформації разом (в одній структурі);
LockFile (), *UnlockFile ()* – блокування і розблокування файлів;
GetFileAttribute (), *SetFileAttribute ()* – встановлення і отримання атрибутів;
GetFileSize() – отримання розмірів файлу;
GetFileTime(), *SetFileTime()* – отримання/встановлення часу створення файлу.

Використання класу Sysinfo

Для того, щоб отримати інформацію про фізичні параметри комп'ютера, можна скористатися класом *Sysinfo.* Для цього:

1) скопіювати файл *Sysinfo.h* в розроблюваний проект;
2) додати в заголовок діалогового класу рядок:
   #include <Sysinfo.h>
3) створити об'єкт *Sysinfo* і викликати функцію-член *TNTGetInfo()*, яка повертає об'єкт типу *CString*, що містить усю інформацію про комп'ютерну систему:

```
Sysinfo m_system;
CString m_content = m_system.TNTGetInfo().
```

А далі використати значення полів цього об'єкта.

Контрольні запитання

1. Розкажіть про методи захисту програм, які базуються на прив'язках.
2. Вкажіть методи доступу до файлової системи комп'ютера.
3. Наведіть засоби для отримання довідкової інформації про дискову систему.
4. Охарактеризуйте засоби для роботи з каталогами файлової системи.
5. Дайте характеристику засобам для пошуку інформації у каталогах. Як це можна використати для побудови системи захисту?
6. Програмні засоби для роботи з файлами (записування, зчитування, позиціювання тощо).
7. Які є засоби для доступу до дескрипторів файлів і які можливості їх зміни?
8. Охарактеризувати методи прив'язки до вінчестера як захист від несанкціонованого копіювання.
9. Які є способи визначення параметрів системи? Дати коротку характеристику кожному з них.
10. Охарактеризуйте вимірювання продуктивності апаратури комп'ютера як метод захисту програм.
11. Які програмні засоби для отримання логічних параметрів комп'ютера ви знаєте?
12. Якими способами можна отримати інформацію про дискову систему комп'ютера?
13. Яким чином можна встановити, отримати та модифікувати атрибути файлів?
14. Як заблокувати (розблокувати) доступ до файлів?
15. Як можна отримати значення фізичних параметрів комп'ютера?
16. Що таке реєстр, яке його призначення і яка його структура?
17. З чого складаються ключі реєстру?
18. Для чого використовують реєстр операційної системи в задачах захисту програмного забезпечення?
19. Які основні програмні функції для роботи з реєстром ви можете навести?
20. Які програмно-апаратні методи захисту ви знаєте? Яким чином здійснюється такий захист?
21. Для чого використовують електронні ключі захисту?
22. Для чого використовуються ключі ідентифікації? Який принцип їх роботи?
23. В чому полягає захист за допомогою довідників?
24. Яким чином можна здійснити прив'язку до кількості запусків програмного забезпечення?
25. Як можна прив'язатися до часу дії програми?

Порядок виконання лабораторної роботи

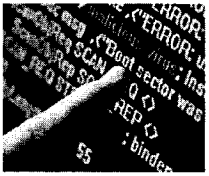
1. Ознайомитись з методами захисту програмного забезпечення за допомогою прив'язки до комп'ютера (див. лекційний матеріал).
2. Дослідити коди програм (див. лаб. роботу № 3):
   - *DevInfo* – програма для отримання відомостей про HDD: серійний номер, модель, версія;
   - *SysInfo* – програма для отримання відомостей про процесор (вид, встановлені прапорці, адреси, …);
   - *Disks* – програма для отримання інформації про логічні диски.
3. Навчитись користуватись API-функціями, що використовуються для отримання інформації про параметри комп'ютера.
4. Дослідити програми, що демонструють роботу з системним реєстром:
   - *CRACK05* – програма прив'язки до часу використання програми;
   - *CRACK09* – прив'язки до кількості запусків програми.
5. Виконати самостійно розробку програми згідно з номером варіанта. Записати в системний реєстр, прочитати з реєстру і вивести на екран таку інформацію:

№ варіанта	Завдання
1	Про логічні диски (кількість, їх логічні серійні номери)
2	Про параметри вінчестера (модель, підмодель і серійний номер)
3	Про параметри вашої «флешки» (назву, серійний номер, файлову систему, системні прапори)
4	Про параметри процесора вашого комп'ютера (тип, кількість, …)
5	Про логічні диски (файлові системи, загальну кількість кластерів)
6	Про вибраний вами файл (назва, час і дата створення, розмір)
7	Про системну директорію і директорію Windows
8	Повні відомості про вибраний вами файл (дату і час останнього запису, дату і час останнього доступу, серійний номер тому-носія, кількість посилань)
9	Повну інформацію про атрибути вибраного файлу у розгорнутому вигляді
10	Список кореневих директорій логічних дисків

6. Оформити звіт з лабораторної роботи.

!!! Обов'язково вміти програмно записати інформацію в системний реєстр і прочитати інформацію з реєстру (для цього можна передбачити у програми два різних режими роботи).

Лабораторна робота № 5



ДОСЛІДЖЕННЯ СТРУКТУРИ ВИКОНУВАНИХ ФАЙЛІВ

Мета і задачі

- Ознайомитись теоретично і на практиці зі структурою заголовків виконуваних файлів.
- Дослідити структуру конкретних виконуваних файлів.
- Навчитись виділяти структурні елементи exe-файлів, аналізувати їх з метою використання при захисті програмного забезпечення шляхом маніпулювання exe-заголовками.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Загальна структура виконуваного файла

В операційних системах, що входять у сімейство Win32, існує декілька типів виконуваних файлів. Ці типи розрізняються за розширеннями файлів, а також за сигнатурою. Сигнатури виконуваних файлів перераховані у заголовному файлі *<winnt.h>* (табл. 5.1).

Таблиця 5.1 – Сигнатури виконуваних файлів

Сигнатура	Значення	Опис
Image_dos_Signature	0x5A4D	MZ
Image_os2_Signature	0x454E	NE
Image_os2_Signature_le	0x454C	LE
Image_VXD_Signature	0x454C	LE
Image_NT_Signature	0x00004550	PE00

Розглянемо лише формат PE-файлів, оскільки саме з цими типами виконуваних файлів ми матимемо справу.

Як правило, після сигнатури файлу розташовується заголовок виконуваного файлу, причому спочатку йде заголовок виконуваного файлу DOS. Чому саме DOS? Тому що виконуваний файл Windows є одночасно і виконуваним файлом DOS. Загальна схема виконуваного PE-файлу така:

old-exe-заголовок – заголовок EXE-файлу DOS
PE File Signature – сигнатура PE-файлу
PE-заголовок – заголовок виконуваного файлу Windows
Таблиця об'єктів (розділів)
Розділи
.....
.....

Заголовки виконуваного файлу

Заголовок DOS

Exe-програми можуть складатися з декількох сегментів (кодів, даних, стека). Exe-файл має заголовок, що використовується при його завантаженні. Заголовок складається з форматованої частини, що містить сигнатуру та дані, необхідні для завантаження Exe-файлу, і таблиці для настроювання адрес (*Relocation Table*). Таблиця складається зі значень у форматі *<сегмент : зсув>*. До зсувів у завантажувальному модулі, на які указують значення в таблиці, після завантаження програми в пам'ять повинна бути додана сегментна адреса, з якої завантажена програма.

Заголовок розташовується на початку файлу EXE-програми. Опис його знаходиться у заголовному файлі *<winnt.h>* і має таку структуру:

```
typedef struct _IMAGE_DOS_HEADER // DOS .EXE header
{
    WORD e_magic;       // сигнатура EXE-файлу (4D5Ah - 'MZ').
    WORD e_cblp;      // довжина останньої сторінки
    WORD e_cp;        // довжина файлу в сторінках (по 512 б)
    WORD e_crlc;      // число елементів у таблиці настроювання адрес
    WORD e_cparhdr;  // довжина заголовка в параграфах (по 200h б)
    WORD e_minalloc; // мінімальний і максимальний обсяг пам'яті,
    WORD e_maxalloc; // яку потрібно виділити після завантаженого
    WORD e_ss;        // зсув сегмента стека (для установки SS)
    WORD e_sp;        // значення SP, установлюване при вході
    WORD e_csum;      // контрольна сума
    WORD e_ip;        // значення IP при вході (стартова адреса)
    WORD e_cs;        // зсув сегмента коду (для установки CS)
    WORD e_lfarlc;    // зсув 1-го ел-та таблиці настроювання адрес
    WORD e_ovno;      // номер оверлея (0 - для кореневого модуля)
      .. .
    LONG e_lfanew;    // зміщення PE-заголовка від початку файлу
  } IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

Якщо значення у слові, яке знаходиться зі зміщенням 18h, більше або рівне 40h, то за зміщенням 3Ch знаходиться значення зміщення, за яким знаходиться сигнатура якогось іншого виконуваного файлу. І якщо ця сигнатура подана значенням «PE00», то файл є виконуваним файлом Windows. Отже, виконуваний файл Windows сам по собі існувати не може.

Таким чином, визначили зміщення на початок виконуваного Windows-файлу. А він також починається із заголовка.

Заголовок виконуваного файлу Windows

У заголовному файлі *<winnt.h>* цей заголовок описаний так:

```
typedef struct _IMAGE_ROM_HEADERS
{     DWORD Signature;       // сигнатура виконуваного файлу
    IMAGE_FILE_HEADER FileHeader; // обов'язкова частина
      IMAGE_ROM_OPTIONAL_HEADER OptionalHeader; // необов'язкова
};
```

Обов'язкова частина заголовка PE-файлу має таку структуру:

```
typedef struct _IMAGE_FILE_HEADER
{     WORD Machine;    // для якого процесора скомпіловано даний код:
                // 0 - невідомий процесор; 0x14C - Intel 386,
                // 0x14D - Intel 486, 0x14E - Intel Pentium і т.д.
      WORD NumberOfSections;      // кількість розділів
      DWORD TimeDateStamp;  // мітка часу створення файлу
      DWORD PointerToSymbolTable;      // зміщення таблиці символів
      DWORD NumberOfSymbols;   // кількість символів у таблиці символів
      WORD SizeOfOptionalHeader;// розмір необов'язкової частини          WORD
Characteristics;// 0x0100 - для 32-бітного комп'ютера,
};                      // 0x1000 - системний файл, 0x2000 - Dll-файл
```

Необов'язкова частина заголовка виконуваного файлу:

```
typedef struct _IMAGE_OPTIONAL_HEADER
{           // Standard fields:
    WORD Magic; // 0x10b
    BYTE MajorLinkerVersion;     // старші і молодші  розряди
    BYTE MinorLinkerVersion;     // номер версії лінкера
    DWORD SizeOfCode; // сумарний розмір всіх розділів
    DWORD SizeOfInitializedData; // розмір ініціалізованих даних
    DWORD SizeOfUninitializedData;// та неініціалізованих даних
    DWORD AddressOfEntryPoint;  // відносна віртуальна  адреса точки
                    // входу у програму (база - у змінній нижче)
    DWORD BaseOfCode;       // відносна адреса програмної секції
    DWORD BaseOfData;       // відносна адреса секції даних
                    // NT additional fields:
    DWORD ImageBase;  // базова адреса файлу відображено у пам'яті
    DWORD SectionAlignment;                  // поля для вирівнювання
    DWORD FileAlignment;
    WORD  MajorOperatingSystemVersion;       // розряди номера старої
    WORD  MinorOperatingSystemVersion;       // версії, яка дозволить
    WORD  MajorImageVersion;                 // виконувати цей файл
    WORD  MinorImageVersion;
    WORD  MajorSubsystemVersion;
    WORD  MinorSubsystemVersion;
    DWORD Reserve1;          // резервні
    DWORD SizeOfImage;       // розмір завантаж. модуля у пам'яті
    DWORD SizeOfHeaders;     // розмір заголовків і таблиці розділів
    DWORD CheckSum;          // контрольна сума виконуваного файлу
    WORD Subsystem;          //тип підсистеми для інтерфейсу
    WORD DllCharacteristics;     // 1 - виклик Dll при першому
          // завантаженні в адресний простір процесу, 2 - інакше
    DWORD SizeOfStackReserve;    // розмір потрібного стека
    DWORD SizeOfStackCommit;
    DWORD SizeOfHeapReserve;
    DWORD SizeOfHeapCommit;
    DWORD LoaderFlags;                  // не використовується
    DWORD NumberOfRvaAndSizes;   // кількість елементів
    IMAGE_DATA_DIRECTORY         // цього масиву
          DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
};
```

Таблиця об'єктів (object table)

Таблиця об'єктів (або таблиця розділів, або таблиця секцій) – сукупність даних певного призначення: про експортовані та імпортовані функції, про ресурси, про переміщення (relocations) і т. д., які компактно розміщені у виконуваному файлі. Іншими словами, таблиця об'єктів – це просто інформація про зміст розділів.

Таблиця об'єктів розміщається відразу ж після заголовка PE-файлу. У заголовному файлі *<winnt.h>* існує опис структури розміром 40 байтів:

```
typedef struct _IMAGE_SECTION_HEADER
{          // заголовок образу розділу
    BYTE Name[IMAGE_SIZEOF_SHORT_NAME];
    union {  DWORD PhysicalAddress;   // розмір розділу
             DWORD VirtualSize;
          } Misc;
    DWORD VirtualAddress;  // зміщення для цього розділу
    DWORD SizeOfRawData;          // округлений розмір розділу
    DWORD PointerToRawData;       // зміщення відносно початку файлу
    DWORD PointerToRelocations; // в exe-файлах не використов.
    DWORD PointerToLinenumbers; // в exe-файлах не використов.
    WORD  NumberOfRelocations;  // в exe-файлах не використов.
    WORD  NumberOfLinenumbers;  // практично не використовується
    DWORD Characteristics; // атрибути розділу (для читання, для
          // запису, чи кешується, чи містить ініційовані дані і т.д.
};
```

Розділи у виконуваному файлі

Розглянемо розділи, які зустрічаються частіше, і те, яку інформацію вони містять.

.text, CODE – секція програмного коду. Ця секція містить код, який генерує компілятор. В компіляторах фірми Microsoft ця секція називається .text, а в компіляторах фірми Borland (нині Inprise) вона називається CODE. З цією секцією нерозривно пов'язані секції імпорту і експорту функцій.

.data, DATA – секція ініціалізованих даних. Назва цієї секції також залежить від виробника: .edata – для компіляторів Microsoft, DATA – для Borland. У даній секції місяться глобальні та статичні змінні, літерали.

.bss – секція неініціалізованих даних. Оскільки ніяких даних у цій секції зберігати не потрібно, то зазвичай її розмір у виконуваному файлі дорівнює 0. Тобто, насправді такої секції у виконуваному файлі немає, а є лише згадування про неї у таблиці секцій. Компілятори фірми Borland цієї секції не створюють.

.idata (ImportDATA) – секція імпортованих функцій. У цій секції знаходяться таблиці, що забезпечують імпорт функцій з бібліотек динамічного компонування.

.edata (ExportDATA) – секція експортованих даних. Секція містить дані про ті функції, які експортуються даним модулем.

.rsrc (ReSouRCes) – секція ресурсів. Вона зберігає всі ресурси, які присутні у виконуваному файлі.

.reloc – секція містить таблицю базових поправок. Вона використовується лише тоді, коли завантажувальник може завантажити файл за адресою, починаючи з якої планував здійснити завантаження редактор зв'язків.

.tls (Thread Local Section). Якщо передбачається використовувати локальну пам'ять потоку, то ці дані не попадають в секції ініціалізованих або неініціалізованих даних. Замість цього вони попадають саме в дану секцію.

У виконуваних файлах зустрічаються й інші секції.

Контрольні запитання

1. Охарактеризуйте структуру виконуваних файлів
2. Що таке заголовки виконуваних файлів, які є їх види?
3. Для чого операційна система використовує заголовки?
4. Що таке таблиця об'єктів (розділів виконуваного файлу)?
5. Які об'єкти (розділи) існують у програмах від різних виробників?
6. Які способи впровадження захисних механізмів у виконувані файли ви можете запропонувати?

Порядок виконання лабораторної роботи

1. Вивчити і зрозуміти структуру виконуваного файлу.
2. Для виконання лабораторної роботи використати результати лабораторної роботи № 2 (Пакувальники) і підготувати файли, які отримані в результаті пакування різними пакувальниками (використати не менше 3-х різних пакувальників). Причому серед цих програм повинні бути програми, розроблені під MS Dos та під Windows.

   У звіт: перелік програм і їх короткий опис.
3. За допомогою програми *Zagolovok*:

– дослідити заголовки виконуваних модулів підготовлених програм;
– дослідити програмні засоби, використовувані для зчитування заголовків;
– *самостійно: прочитати і видати на екран таблицю розділів файлу.

   У звіт: результати дослідження з поясненнями.
4. За допомогою програм *Hiew* та *PE Explorer* проаналізувати структуру підготовлених виконуваних файлів. Результати дослідження можна оформити у вигляді таблиці.

Програма	Пакувальник	Заголовок		Розділи файлу	Спосіб упакування	Можливості використання для захисту
		Dos-заголовок	PE-заголовок			
...	...	...	...	...	...	...

Лабораторна робота № 6

ДОСЛІДЖЕННЯ РОБОТИ ДЕКОМПІЛЯТОРІВ ТА РЕДАКТОРІВ РЕСУРСІВ

Мета і задачі

- Ознайомитись на практиці з такими інструментами статичного дослідження як декомпілятори.
- Виконати декомпіляцію програм, написаних за допомогою Visual C++, Delphi, Visual Basic, Java та інших.
- Дослідити на практиці роботу редакторів ресурсів.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Загальні відомості про компілятори і декомпілятори

Із зростанням потужності ЕОМ досить широке поширення набули компілятори, що створюють не «чистий» машинний код, а деякий набір умовних інструкцій, який виконується за допомогою інтерпретатора. Інтерпретатор може як поставлятися окремо (Java), так і бути «прикріпленим» до самої програми (хоча формально не інтерпретатор прикріпляється до програми, а програма до інтерпретатора. Прикладом може слугувати Visual Basic в режимі компіляції в p-code). Інтерпретаторами є практично всі інсталятори (у їх основі лежить інтерпретатор інсталяційного скрипту, хоча сам процес створення такого скрипту може бути прихований за допомогою візуальних засобів). Та й «звичайні» компілювальні мови можуть створювати код, прямий аналіз якого вельми складний.

Для аналізу таких програм використовуються спеціалізовані утиліти, що переводять код, зрозумілий лише інтерпретатору, у форму, зручнішу для розуміння людиною. Деякі декомпілятори можуть витягувати інформацію про елементи інтерфейсу, створені візуальними засобами. У будь-якому випадку, не потрібно чекати від декомпіляторів відновлення початкового тексту програми. Якщо програма, що декомпілює, успішно компілюється і зберігає дієздатність – це виключення, а не правило.

Складність коду, що генерується компіляторами мов високого рівня, використання віртуальних машин, нові формати для зберігання даних – VCL, Visual Basic, Java, FoxPro, .NET та ін. – все це призводить до того, що звичних дизасемблерів native-коду вже не вистачає для того, щоб хоч якось розібратися в роботі програми. Реверсеру потрібні нові інструменти, які змогли б розкопати в мільйонах байтів досліджуваної програми знайомі рядки мови, на якому вона була написана. Таким чином, іноді необхідні декомпілятори.

Відновлення початкового коду програми багато в чому стало можливим через масу зайвої інформації про оригінальний код у файлі, що компілюється, а також через однотипність структур і операторів, використовуваних в мовах високого рівня. Декомпілятори аналізують надмірну інформацію, і, знаючи, як компілятор тієї або іншої мови подає деякі структури, намагаються створити деяку подібність первісного коду.

Краще всього декомпілюються програми, які виконуються не напряму процесором, а віртуальною машиною (написані на Java, Visual Basic, FoxPro, .NET і т. п.). Причина цього криється у тому, що інструкції віртуальних машин, як правило, високорівневі і об'єднують відразу декілька машинних команд. Іншими словами, це ті ж самі оператори і ключові слова мови програмування, тільки записані дещо інакше, з деякою оптимізацією інструкцій мови розробки.

Другими за простотою декомпілювання йдуть програми, написані на таких мовах програмування як Delphi і C++ Builder. Хоча ці мови не мають справи з віртуальними машинами (окрім хіба що платформи .NET), а компілюють програми в нормальний native-код, вони люблять використовувати власні стандартні бібліотеки на зразок VCL і залишають в EXE-файлах багато зайвої інформації, використовуючи яку можна також цілком успішно відновити початковий код.

Що ж до сучасного середовища розробки .NET, то завдяки дуже великій кількості інформації, що зберігається в exe-файлах скомпільованих програм, можна майже зі 100% точністю відновити початковий код, написаний кодером. Але дуже часто розробники проектів у середовищах розробки .NET використовують так звані обфускатори коду, здатні вичищати зайву інформацію з програм, щоб хоч якось перешкоджати декомпілюванню. Про обфускатори і їх можливості поки що мало хто знає, а ось саме середовище розробки використовується вже щосили. І це призводить до зростання кількості кейгенів.

Декомпілятори

Декомпілятори програм з мови Delphi

Dede by DaFixer. Найзнаменитіший декомпілятор Delphi. Працює з програмами, скомпільованими будь-якими версіями Delphi, окрім восьмої (оскільки вона створює .NET-код). Мало того, що це досить корисний інструмент, він ще й має відкриті для програмістів тексти початкового коду однієї з його старих версій. Іноді це може стати в нагоді тому, хто вивчає код, який генерується Borland-програмами.

Ця програма має багато можливостей:

- надає всі форми в оригінальному вигляді;
- дає можливість «полазити» по процедурах і функціях, що є в програмі;

– крім дизасемблерного лістингу цих функцій, програма намагається розпізнати стандартні оператори і типи Delphi та додає їх в коментарі до асемблерного коду;
– може розпізнавати виключні ситуації, тобто блоки типу «try-except»;
– може створювати код програми, який можна відкрити в Delphi.

Загалом, досить потужній декомпілятор. Головний *недолік* Dede у тому, що він не вміє «висмикувати» з exe-файлу компоненти, використані в програмі. Через це у згенерованому коді програми присутня безліч нерозпізнаних типів даних. Примусити одержаний код працювати все одно не вдасться, а досліджувати – будь-ласка.

SourceRescuer *(www.ems-hitech.com).* Ще один декомпілятор Delphi, але простіший. Уміє відновлювати форми і генерувати заголовки pas-файлів. Головна відмінність від Dede – миттєва робота і більш ергономічний інтерфейс. З головних особливостей даного декомпілятора можна виділити те, що він може створювати шаблон програмного коду не тільки у форматі Delphi, але й у форматі Builder'a. Це пояснюється просто – після компіляції програми Delphi і C++ Builder мало чим відрізняються. Розповсюджується в двох видах: GUI і консольному. Потребує реєстрації.

Декомпілятори програм з мов C++ і Delphi

REC by Giampiero Caprino *(www.backerstreet.com/rec/recdload.htm).* Повна назва – the Reverse Engineering Compiler. Програма, яка з успіхом перетворить у початковий код на C будь-який виконуваний файл. Звичайно, код програми виходить мало схожим на оригінал, але розібратися в ньому нескладно. REC визначає функції досліджуваного файлу, основні блоки і структури мови, такі як *if, for, switch*, виклики API, і робить взагалі все можливе, щоб код став зрозумілим С-програмісту. Проте часто можна у згенерованому коді побачити щось на зразок:

```
eax++;
for(ecx=1000;ecx!=0;ecx--)
{
  ebx = ebx&ecx;
}
```

Це не одразу зрозуміло, але нормально. І, мабуть, краще досліджувати це, ніж:

```
00400000: inc eax
00400001: mov ecx, 1000
00400006: and ebx, ecx
00400008: loop 00400006
```

Але це залежить від кваліфікації програміста.

З приємних дрібниць даної програми можна відзначити:

– кросплатформеність (окрім Windows, декомпілятор вважає рідними Linux, Mac OS X і Solaris);
– підтримку декількох форматів виконуваних файлів (REC не обмежився лише PE і COFF, він уміє аналізувати і ELF, і AOUT, і навіть Playstation PS-X).

Декомпілятори .NET-збірок

.NET Reflector by Lutz Roeder *(www.aisto.com/roeder/dotnet).*

Це потужний декомпілятор .NET-збірок, який є безкоштовним і таким, що динамічно розвивається. Практично будь-яку програму, зібрану за новою технологією, він без проблем подасть у вигляді повного програмного коду зі всім деревом успадкування класів. Підсвічування синтаксису, гіперпосилання на об'єкти класів, зручний і приємний інтерфейс – все це говорить лише про одне: у декомпілюванні .NET-збірок цьому інструменту немає рівних. Якщо треба дослідити .NET-програму або просто поглянути на свою розробку очима реверсера, це чудовий програмний продукт.

Декомпілятори класів у Java

Decafe Pro*(http://decafe.hypermart.net).* Декомпілятор Java, який реконструює код першоджерела від скомпільованих двійкових файлів класу (аплети). Decafe може декомпілювати найскладніші аплети і відтворити точний початковий код. Хоча згенерований код може містити неточності або помилки.

Decafe – це умовно безкоштовний продукт (as-is). Платня за пересилку становить $29 США.

DJ Java Decompiler. Декомпілятор Java-класів. Досить простий і зручний. Відкриваючи в ньому клас, одразу ж бачимо його програмний код. Є непогане підсвічування синтаксису, пошук і настроювання. Також є браузер класів і об'єктів. Загалом, дуже непоганий і цікавий декомпілятор. До недоліків можна віднести те, що exe-файли, написані на Java, не декомпілює.

Декомпілятори Visual Basic

Злом додатків, створених за допомогою Visual Basic і скомпільованих в p-код, був мукою для зламників, поки не стали з'являтися нормальні декомпілятори цієї мови. Зараз їх вже багато. Вони діляться на три типи:

- декомпілятори форм;
- редактори форм;
- декомпілятори p-коду.

Є програми, що поєднують в собі відразу декілька можливостей.

VBRezQ *(www.vbrezq.com).* Один з найстабільніших декомпіляторів форм. Хоча, окрім стабільності, він нічим більше і не примітний. Оголошення API-функцій робить без параметрів, від чого користі мало. Код не декомпілює взагалі. Має досить докладну документацію, але й коштує досить дорого. Загалом, його, напевно, варто використовувати, якщо потрібно декомпілювати тільки елементи інтерфейсу.

VB Editor by HEXMAN *(www.multimania.com/hexman).*Це абсолютно безкоштовний редактор форм і розташованих на них об'єктів. Якщо хтось займається українізацією програм, але не може нічого зробити з

тими програмами, що написані на VB (Restorator тут безсилий), то тут допоможе VB Editor. На основі зроблених змін редактор може згенерувати форму, тому він цілком згодиться як гідна альтернатива згаданому вище VBRezQ.

VBReFormer by Sylvain Bruyere *(www.decompiler-vb.tk).* Ще одна робота французьких програмістів. Ця програма потужніша за попередню. Крім перегляду і зміни форм, надає також можливість видобути з exe'шника зображення, які звичайно знаходяться в frx-файлах. Trial-версія дозволяє тільки проглядати результати. При цьому не тільки забороняється зберігати результат, але і постійно онулюється буфер обміну, щоб виключити можливість копіювання. Звідси, перш ніж запускати цю програму, потрібно переконатися, що в буфері немає потрібних, але ще не збережених даних. Крім всього іншого, VBReFormer надає для огляду деякі дані із заголовка exe-файлу, тому адресу точки входу в програму і Image Base можна взнати без зусиль. Ще один плюс програми в тому, що вона вміє розпізнавати використовувані в програмі ActiveX-файли і дозволяє проглянути всі їх властивості та методи. Шкода тільки, що не використовує цю інформацію при генерації форм – там всі ActiveX-файли виглядають дещо убого, без властивостей і привласнених їм даних.

При бажанні ця програма може «оббігти» весь жорсткий диск у пошуках програм, написаних на VB. Це потрібно, напевно, для тестування можливостей на різних exe'шниках.

RaceEx6 *(www.raceco.us).* Ця програма намагається декомпілювати і форми, і p-код, але подає на екран всю інформацію у такому убогому вигляді, що не зрозумієш, які дані до чого належать. Так само як і попередня програма, уміє «висмикувати» графіку з програм, написаних на VB. Коли працює над p-кодом, декомпілює тільки методи – з переданими в них параметрами не справляється. Загалом, якби не деякі проблеми з інтерфейсом, то можна було б вважати цю програму цілком нормальною і дієздатною.

Exdec by josephco *(www.wasm.ru).* Напевно найвідоміший декомпілятор p-коду. Можливість у програми всього одна – декомпілювати p-код у тому вигляді, в якому він є. Тобто того коду, який писав програміст на VB, не побачиш, а ось те, що згенерував компілятор – так, причому в досить читабельному для професіонала вигляді.

VBParser. Майже повний аналог ExDec, тільки написаний китайськими розробниками. Результат своєї роботи не тільки виводить на екран, але і зберігає у файлі ParseVB.txt. На випадок її падіння (а таке частенько трапляється) цей файл дуже стане в нагоді.

P32Dasm by DARKER. Краща альтернатива ExDec'у і VBParser'у. На відміну від двох попередніх, написана на VB (ExDec і VBParser – на C++) і достатньо непогано розвивається останнім часом. Має підсвічування синтаксису, калькулятор адрес, уміє декомпілювати з певного зсуву у

файлі, а також, подібно дизасемблерам, здатна подати всі рядки і функції, використовувані в програмі, в зручному списку з можливістю миттєво перейти на потрібну позицію в лістингу. Головний недолік – нестабільність роботи і повільна швидкість, в іншому ж програма досить цікава. Розповсюджується безкоштовно.

VBDE by iorior. Досить непоганий декомпілятор, причому безкоштовний. Декомпілює форми (правда, без ActiveX). Видає адреси на всі процедури, а якщо це можливо, то виводить не тільки адресу процедури, але і її ім'я, що значно спрощує аналіз. Намагається декомпілювати native-код, проте, в більшості випадків, окрім операторів додавання, віднімання і виведення MessageBox'а нічого декомпілювати не може. Не дивлячись на всі ці недоліки, цей продукт досить зручний і стабільний.

VB Decompiler by GPcH. Це декомпілятор VB. Має багато можливостей, при цьому інтерфейс не має нічого зайвого, не захаращений тим, що ніколи не стане в нагоді. А можливості такі:

- декомпілятор форм з підтримкою ActiveX'ів (при цьому декомпілюються тільки загальні для всіх ActiveX'ів властивості);
- декомпілятор p-коду (причому програма намагається перевести p-код у подібний до початкового читабельний вигляд);
- декомпілятор посилань на API (при цьому вони записуються вже в об'явленому вигляді із всім списком параметрів);
- підсвічування синтаксису;
- для кожного модуля з кодом є свій список символьних рядків, з можливістю миттєво перейти на ділянку коду їх використання;
- є пошук, що допомагає знайти потрібний код в активному вікні;
- результат роботи можна зберігати, причому разом з кодом і формами зберігаються і frx-файли з графікою і коректно прописуються посилання на ці графічні об'єкти у формах.

Даний декомпілятор призначений для відновлення початкових кодів, якщо раптом їх втрачено і є тільки EXE.

Висновок. Як бачимо, дефіциту декомпіляторов не спостерігається. При цьому дуже добре відчувається різниця між професійними і любительськими розробками, вона велика як в якості, так і в ціні. Не дивлячись на це, багато хто зможе обійтися безкоштовними розробками.

Пропонується самостійно знайти в Інтернеті програми для декомпіляції виконуваних модулів різних мов програмування, випробувати їх і зробити порівняльний аналіз якості декомпіляції.

Модифікація програм за допомогою редакторів ресурсів

Більшість незареєстрованих версій характеризуються тим, що частина їх можливостей заблокована. Якщо програма передбачає реєстрацію, то великих проблем при зломі не виникає. Зовсім інша справа, коли реєстра-

ція не передбачена і в нашому розпорядженні DEMO-версія з обмеженими можливостями. Інакше кажучи, є дві програми, не пов'язані між собою, – повна і демонстраційна версії. Дуже вірогідно, що злом останньої виявиться неможливим, оскільки код, що виконує деякі функції, в програмі фізично відсутній.

Проте найчастіше код все ж таки фізично присутній, але не одержує управління. Наприклад, просто заблоковані деякі пункти меню. Таке дійсно зустрічається дуже часто і легко програмується. Все, що треба зробити програмісту, – це помітити в редакторі ресурсів деякі елементи управління або меню як 'Disabled'. І ось що роблять хакери у цьому випадку.

Те, що просто робиться, так само просто і ламається. Необхідно скористатися будь-яким редактором ресурсів. Наприклад, можна користуватися Symantec ResoLirceStudio, проте придатний і будь-який інший. Завантажимо в нього наш файл. Подальші дії залежать від інтерфейсу вибраної програми і не повинні викликати ускладнень, за винятком тих ситуацій, коли вибраний редактор не підтримує використовуваного формату ресурсів або некоректно працює з ними. Наприклад, за допомогою Resource Workshop деякі операції не вдається виконати. Він іноді необоротно псує ресурс діалогу, хоча з розблокуванням меню справляється добре. Отже, часто використовується декілька редакторів. Треба просто набути певного досвіду, щоб знати, коли яким редактором ресурсів користуватися.

Запустимо на виконання Crack0D. Бачимо, що один з пунктів меню (Help2) заблокований (рис. 6.1).

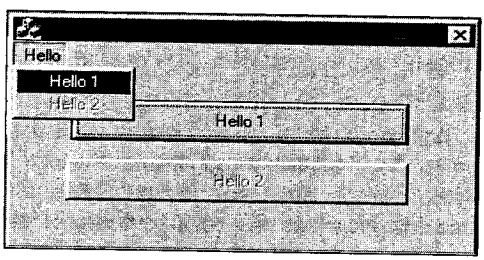


Рисунок 6.1 – Вікно програми Crack0D

Щоб розблоковувати елементи управління або меню, необхідно викликати властивості об'єкта і зняти позначку *Disabled* або *Grayed*, після чого зберегти зміни (рис. 6.2).

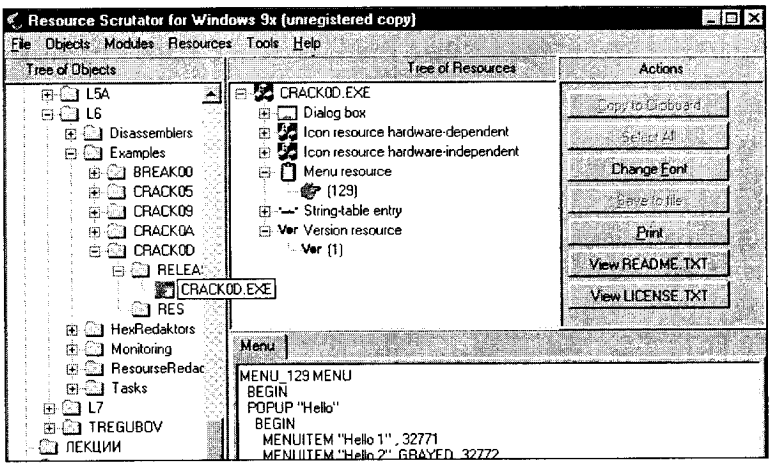


Рисунок 6.2 – Вигляд вікна редактора ресурсів

Запустимо програму, щоб перевірити нашу роботу. Не виправивши жодного байта коду і навіть не вдаючись до допомоги дизасемблера і налагоджувача, ми здійснили злом. Дивно, що такі захисти зустрічаються дотепер, і не так вже рідко. Зрозуміло, що немає нічого простіше і приємніше, ніж редагувати ресурси у виконуваному файлі, тому дехто вдається до явних викликів API типу *EnableWindow* (*false*). Тобто блокуються елементи управління безпосередньо під час роботи. Зрозуміло, можна перехопити цей виклик налагоджувачем і видалити захисний код. Саме так зробить будь-який хакер і навіть крекер. Пересічний же користувач зупинить свій вибір на програмі, подібній Custornizer, яка дозволяє «на льоту» міняти властивості будь-якого вікна, а згодом робити це і автоматично.

У даному прикладі показана робота редактора ресурсів Resource Scrutator, але в мережі Інтеренет є лише демо-версія цієї програми. Для виконання лабораторної роботи надаються інші редактори ресурсів.

Висновок. Отже, необхідно підсилити реалізацію захисту даного типу, щоб її розкриття не було доступне широкому колу користувачів – не розміщати у програмі дані, що мають бути недоступними у незареєстрованій версії програми, у явному вигляді.

Досить ввести деяку змінну типу *Registered* і перевіряти при натисненні на кнопку це значення. Якщо *Registered* дорівнює нулю, а користувач якимось загадковим чином все ж таки ухитрився натиснути заблоковану кнопку, то повторно блокуємо кнопку або завершуємо роботу, мотивуючи це несанкціонованими діями.

Контрольні запитання

1. Що таке статичне та синтаксичне дослідження програм?
2. Які інструменти статичного дослідження програм ви знаєте?
3. Для чого слугують декомпілятори?
4. Від чого залежить якість роботи декомпіляторів?
5. Наведіть приклади декомпіляторів різних мов і охарактеризуйте їх.
6. Обгрунтуйте відмінності у роботі різних декомпіляторів, здійснивши їх порівняльний аналіз.
7. Для чого використовують редактори ресурсів?
8. У яких випадках робота редакторів ресурсів є задовільною, а у яких – ні? Чому?
9. Наведіть відомі вам редактори ресурсів.

Порядок виконання лабораторної роботи

1. Ознайомитись з теоретичними відомостями щодо порядку статичного вивчення програм програм.
   У звіт:
   - поняття декомпіляторів та їх призначення;
   - поняття редакторів ресурсів та їх призначення;
   - поняття p-коду і native-коду, їх відмінності.
2. Знайти в Інтернеті дієздатний декомпілятор з мови C/C++ (наприклад, recStud), декомпілювати програми, написані мовою C++: програму Example; програму Prog1.
   У звіт:
   - характеристика декомпілятора, його інтерфейс та можливості;
   - назва програми-об'єкта, її структура;
   - оцінити якість декомпілювання, порівнюючи згенерований код із початковим (для цього вибрати фрагмент початкового коду і знайти відповідний фрагмент у згенерованому декомпілятором).
3. Знайти в Інтернеті дієздатний декомпілятор з мови Java (наприклад, DecafePro), декомпілювати класи, створені інтерпретатором Java: Clock; TicTacToe; ScrollText (можна взяти власні програми).
   У звіт:
   - характеристика декомпілятора, його інтерфейс та можливості;
   - назва програми-об'єкта, її структура;
   - оцінити якість декомпілювання, порівнюючи згенерований код із початковим (для цього вибрати фрагмент початкового коду і знайти відповідний фрагмент у згенерованому декомпілятором).
4. Знайти в Інтернеті дієздатний декомпілятор з мови Visual Basic (наприклад, VbDecompiler), декомпілювати програму kurs, створену у Visual Basic (можна взяти іншу програму).

У звіт:

- характеристика декомпілятора, його інтерфейс та можливості;
- назва програми-об'єкта, її структура;
- оцінити якість декомпілювання, порівняти якість декомпіляції програм, відкомпільованих у native-code і p-code.

5. Знайти в Інтернеті дієздатний декомпілятор з мови Delphi (наприклад, DeDe) на прикладі програми kursova (або іншої).

   У звіт:

   - характеристика декомпілятора, його інтерфейс та можливості;
   - назва програми-об'єкта, її структура;
   - оцінити якість декомпілювання, порівняти якість декомпіляції програм, відкомпільованих у native-code і p-code.
6. Знайти в Інтернеті 2–3 дієздатні програми-редактори ресурсів і ознайомитись з ними, дослідити їх можливості.
7. За допомогою кожного з редакторів ресурсів дослідити програми-об'єкти (наприклад, курсова робота II курсу), внести деякі зміни і створити новий виконуваний файл.

   У звіт:

   - назва, можливості, інтерфейс програми-редактора ресурсів;
   - назва програми-об'єкта, її призначення, використані ресурси;
   - внесені зміни (навести вигляд програми до змін і після);
   - результати здійснених змін (чи залишилася програма дієздатною, чи змінено демо-версію тощо).
8. Сформулювати висновки по лабораторній роботі:
   - рекомендації щодо покращення захисту з метою боротьби з декомпіляторами;
   - рекомендації по розробці програм з метою унеможливлювання їх дослідження і модифікації під управлінням редакторів ресурсів.

Лабораторна робота № 7

СТАТИЧНЕ ДОСЛІДЖЕННЯ ПРОГРАМ. ДИЗАСЕМБЛЮВАННЯ І МОДИФІКАЦІЯ

Мета і задачі

- Ознайомитись на практиці з такими інструментами статичного дослідження, як дизасемблери.
- Дослідити можливості різних дизасемблерів.
- Ознайомитись з роботою шістнадцяткових редакторів.
- Навчитись досліджувати виконувані модулі програм і вносити зміни у їх код за допомогою шістнадцяткових редакторів.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Інструменти для статичного дослідження

Дизасемблери та шістнадцяткові редактори є головними засобами статичного дослідження програм. Традиційно обидва ці типи інструментів використовуються в парі. Дизасемблер видає «чистий код», хоча сучасні дизасемблери здатні також розпізнати виклики стандартних функцій, виділити локальні змінні в процедурах і надають інший сервіс. Користуючись дизасемблером, можна здогадуватися про те, які дані одержує певна функція, які параметри і що вони означають; щоб з'ясувати це, найчастіше потрібне вивчення якщо не всієї програми, то досить значної її частини.

Дизасемблери являють собою досить потужний інструмент, який може виявитися ефективним при зломі програм. Правильно спроектований захист повинен передбачати спробу дизасемблювання коду і протидіяти їй.

Початковими даними для дизасемблера є програма або окрема ділянка коду. Результатом його роботи звичайно є лістинг (початковий текст програми мовою асемблера), який, в ідеалі, повинен бути максимально близьким до оригіналу. Дизасемблер відновлює код програми, послідовно декодуючи команду за командою з того місця, куди програмі передається управління при її запуску. Він намагається дотримуватись порядку виконання інструкцій, відрізняючи їх таким чином від даних.

Потрібно зазначити, що дизасемблювання саме по собі проблематичне навіть у тому випадку, коли протидія налагоджувачу не передбачена. Річ у тому, що в комп'ютері дані і код знаходяться в єдиному адресному просторі, через це задача автоматичної ідентифікації якоїсь ділянки програми як даних або коду досить складна і неоднозначна: програміст може використовувати код як дані або навпаки. Виходячи з цього, можна стверджувати, що асемблювання і лінкування програми – процес необоротний, і

одержати точний початковий текст програми в загальному випадку неможливо.

Шістнадцяткові редактори – це, поза сумнівом, найдавніші з інструментів програмістів, які ведуть свою славну історію з тих часів, коли програмісти ще вміли «читати» і правити виконуваний код, не вдаючись до допомоги дизасемблерів. Таємне мистецтво читання коду ще збереглося у віддалених куточках світу, але останніми роками практично втратило актуальність. І тепер, в основному, лише крекери практикують цей містичний обряд, виправляючи в «неправильних» програмах одні коди на інші, змінюючи структуру програми, порядок переходів і т. і.

Дослідження програм, захищених паролем

Приклад 1.

Розглянемо програму ***BREAK00***, в якій пароль зберігається в тілі програми. Це програма під Win32, яка використовує MFC. Запустимо break00.exe. Програма просить ввести пароль (рис. 7.1).

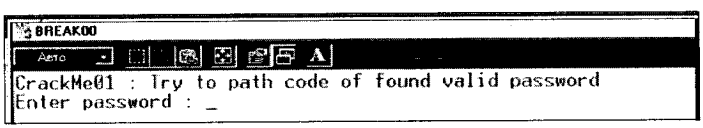


Рисунок 7.1 – Вигляд вікна програми при запуску

Для того, щоб знайти правильний пароль, треба просто проглянути дамп програми і пошукати всі текстові рядки, які можуть бути паролем.

Очевидно, що Break00 не є добре захищеною програмою. У ній немає ні шифрованого коду, ні «пасток» для дизасемблерів. SOURCER, IDA або будь-який інший дизасемблер легко справиться з цією задачею.

Безперечно, серед існуючих на сьогоднішній день дизасемблерів одним з кращих є IDA. Особливо ідеально він підходить для злому і вивчення захищених програм. Ним і скористаємося.

Запустимо дизасемблер на виконання за допомогою послідовності команд *File>OpenFile*, завантажимо наш досліджуваний файл. В результаті дизасемблер видасть кілометровий лістинг. А нам треба знайти в програмі те місце, де знаходиться оригінальний пароль. Звичайно, можна проглядати весь наданий лістинг (а якщо це програма не на 5–10 операторів, а на сотні і тисячі?). Навіть для такої маленької програми це не так просто.

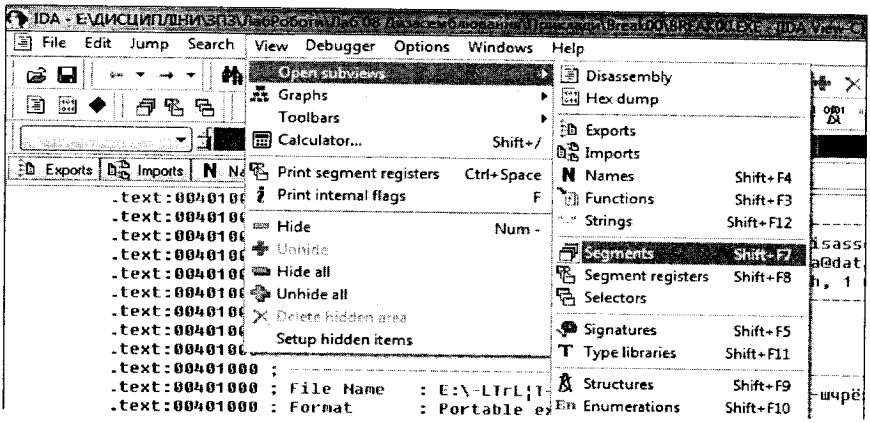
Для того, щоб локалізувати місце пошуку, скористаємося меню. Викличемо послідовність команд *View→OpenSubviews→Segments* або *Shift+F7* (рис. 7.2, а).

В результаті дизасемблер надасть нам список розділів (сегментів) програми (рис. 7.2, б).

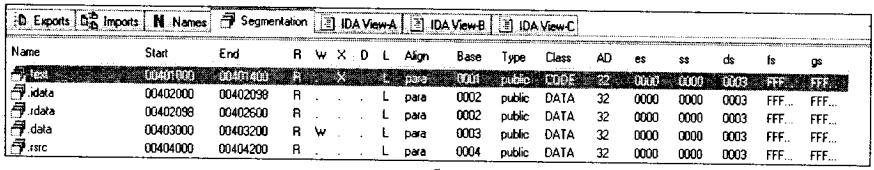
Припускаємо, що після введення пароля з клавіатури в програмі повинно відбуватися порівняння оригінального пароля з введеним. Отже,

оригінальний пароль повинен якось зберігатися в програмі. Очевидним способом є просте посимвольне порівняння типу:

```
if ((s0-ch) !=«XXXXX») cout<<endl<<«Password fail»<<endl;
```



а



б

Рисунок 7.2 – Вигляд вікна таблиці розділів

Після запуску програми ми бачили запрошення *«Enter password:»*. Отже, нам потрібно шукати в лістингу саме цей рядок або рядок *«Password fail»* і зрозуміти, чому ми сюди попали і не виконали програму. А вже недалеко від цього рядка може розташуватися і сам пароль. Якщо припустити, що порівняння рядків відбувається безпосередньо в програмі, то в таблиці розділів потрібно вибрати секцію *.text*. Хоча цілком можливо, що даний рядок може бути записаний в області даних, і тоді треба шукати його в секції .data. В даному випадку ми локалізували область пошуку, вибравши у вікні таблиці розділів секцію *.text* (секція коду програми).

Звичайно, можна проглядати весь лістинг у пошуках необхідного рядка, але це досить трудомістка процедура. В загальному випадку цей пошук потрібного місця здійснюється декількома способами:

1) скористаємося для цього послідовністю команд для пошуку символьного рядка *Alt-T (Search-Text)* (рис. 7.3). Вводимо текст шуканого рядка і бачимо, що недалеко від нього знаходиться і рядок «*KPNC*». Можливо він і є паролем. Пробуємо. Результат позитивний;

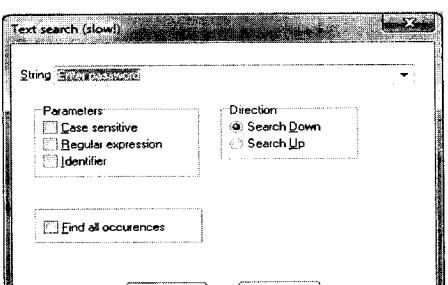


Рисунок 7.3 – Вигляд вікна пошуку

2) відкрити вікно *Name Window* і спробувати знайти у ньому потрібний текст (рис. 7.4);

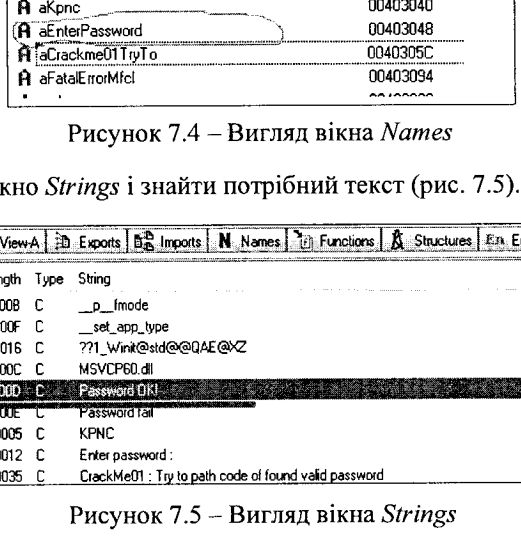


Рисунок 7.4 – Вигляд вікна *Names*

3) відкрити вікно *Strings* і знайти потрібний текст (рис. 7.5).

Рисунок 7.5 – Вигляд вікна *Strings*

Клацнувши на потрібний рядок, дизасемблер показує нам, де цей рядок зустрічається вперше (рис. 7.6).

```
.data:0040301B                  db    0
.data:0040301C                  db    0
.data:0040301D                  db    0
.data:0040301E                  db    0
.data:0040301F                  db    0
.data:00403020 aPassword0k      db 'Password OK!',0     ; DATA XREF: _main+10A↑o
.data:0040302D                  align 10h
.data:00403030 aPasswordFail    db 'Password fail',0    ; DATA XREF: _main+F8↑o
.data:0040303E                  align 10h
.data:00403040 aKpnc            db 'KPNC',0             ; DATA XREF: _main+D5↑o
.data:00403045                  align 4
.data:00403048 aEnterPassword   db 'Enter password : ',0 ; DATA XREF: _main+AC↑o
.data:0040305A                  align 4
```

Рисунок 7.6 – Вигляд потрібної команди у сегменті даних

Ми бачимо, що як коментар надано адресу, де цей рядок зустрічається у програмі. Навівши на цю адресу курсор миші, бачимо підказку – вигляд фрагмента програми і сегмента коду, де використовується дана змінна (рис. 7.7).

```
.data:0040301F                  db    0
.data:00403020 aPassword0k      db 'Password OK!',0     ; DATA XREF: _main+10A↑o
                push    eax
                call    ds:?endl@std@@YAAAV?$basic_ostream@DU?$char_traits@D@std@@@1@AAV21@@Z ; std::en
                add     esp, 10h
                jmp     short loc_4010E7
;

loc_401144:                             ; CODE XREF: _main+E8↑j
                call    ds:?endl@std@@YAAAV?$basic_ostream@DU?$char_traits@D@std@@@1@AAV21@@Z ; std::en
                push    offset aPassword0k ; "Password OK!"
                push    eax
```

Рисунок 7.7 – Вигляд контекстної підказки

Бачимо, що попали саме туди, куди треба. Натиснувши *Enter* або відкривши контекстне меню, знайшовши опцію *Jmp to operand*, переходимо у цей фрагмент коду в сегменті коду (рис. 7.8).

```
.text:00401122                  mov     eax, ds:?cout@std@@3V?$basic_ostream@DU?$char
.text:00401127                  push    eax
.text:00401128                  jz      short loc_401144
.text:0040112A                  call    ds:?endl@std@@YAAAV?$basic_ostream@DU?$char_t
.text:00401130                  push    offset aPasswordFail ; "Password fail"
.text:00401135                  push    eax
.text:00401136                  call    esi ; std::operator<<(std::basic_ostream<char
.text:00401138                  push    eax
.text:00401139                  call    ds:?endl@std@@YAAAV?$basic_ostream@DU?$char_t
.text:0040113F                  add     esp, 10h
.text:00401142                  jmp     short loc_4010E7
.text:00401144 ; ---------------------------------------------------------------------
.text:00401144
.text:00401144 loc_401144:                              ; CODE XREF: _main+E8↑j
.text:00401144                  call    ds:?endl@std@@YAAAV?$basic_ostream@DU?$char_t
.text:0040114A                  push    offset aPassword0k ; "Password OK!"
.text:0040114F                  push    eax
```

Рисунок 7.8 – Вигляд потрібного фрагмента коду

А далі, локалізувавши місце, де може відбутися перевірка пароля, запам'ятовуємо адресу потрібної команди і змінюємо у шістнадцятковому редакторі на потрібний код.

Взагалі-то, ми не зовсім правильно вчинили, що стали одразу ж пробувати ввести правильний пароль. Адже цілком може бути, що розробник, припускаючи такі наші дії, підсунув «помилковий» пароль, який, замість того, щоб запустити програму, навпаки, видалить це з диска або, що ще гірше, відформатує сам диск. Звичайно, такі прийоми не належать до числа «красивих» і не так вже часто зустрічаються, щоб примусити нас приймати цю загрозу в розрахунок, але аналіз використання результатів порівняння дозволить нам змінити код так, щоб програма взагалі не запитувала пароль або, в решті решт, просто сприймала будь-який як правильний. Тому в таких випадках потрібно проаналізувати код програми глибше.

Висновок. Якщо вже ми і вирішили захищати свій програмний продукт паролем за допомогою порівняння двох рядків в програмі, то потрібно ретельніше продумати алгоритм захисту і принаймні не розташовувати фрагмент програми, в якому пароль запитується, і фрагмент, де він порівнюється з оригінальним, близько один від одного, а розносити їх за програмою. І крім того, у відкритому вигляді зберігати пароль у програмі не потрібно – треба хоча б хешувати його.

Статичне дослідження програм, обмежених часом використання

Одним з популярних обмежень DEMO-версій є обмежений час використання. Бувають принаймні два види обмежень:

- відлік часу йде від моменту першого запуску;
- програма працює до деякої наперед встановленої дати.

Зрозуміло, перше набагато зручніше, але і більш уразливо, оскільки необхідно десь зберегти дату першого запуску (причому переконатися, що він саме перший). Є дуже небагато способів це зробити. Практично розробники обмежені реєстром або зовнішнім файлом. Змінювати код самої програми неприпустимо, оскільки це викличе протест з боку антивірусів, а значить, і з боку використовуючих їх клієнтів. Під MS-DOS програми минулого покоління могли записувати в інженерні циліндри жорсткого диска, в невикористаний кінець останнього кластера файлу, невикористані поля CMOS. Сьогодні ситуація змінилася. Сучасні ОС типу Windows NT взагалі не дадуть непривілейованому користувачу прямого доступу до диска. Йде активне впровадження мережевих технологій, а отже, захисний механізм повинен успішно функціонувати і на мережевій машині. Таким чином, практично єдиною відповідною кандидатурою виглядає реєстр. Проте всі звернення до нього дуже легко відстежити і відредагувати. Або можна переустановити ОС, знищивши реєстр.

Втім, не менш уразлива ця технологія по відношенню до переведення системної дати, що доступно навіть некваліфікованим користувачам.

Проте робота з некоректною датою викликає певні незручності, а в деяких випадках навіть недопустима, тому переважно все ж таки модифікувати код програми, прибравши обмеження за часом. Або принаймні відстежити збереження моменту першого запуску і підредагувати його. Друге часто значно простіше, тому почнемо з нього.

Приклад 2.

Розглянемо для прикладу програму **crack05.exe**. Програма при першому запуску запам'ятовує поточну дату (рис. 7.9), щодня при запуску програми перевіряє, чи закінчився час використання (рис. 7.10), і після закінчення 20 днів з цієї миті припиняє роботу (рис. 7.11).

Рисунок 7.9 – Вигляд вікна програми при першому запуску

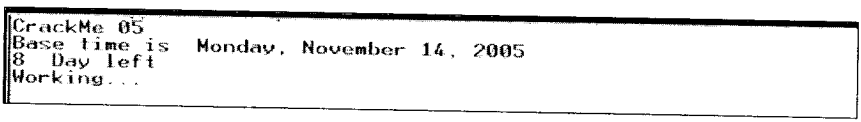


Рисунок 7.10 – Вигляд вікна програми при успішному спрацьовуванні захисту

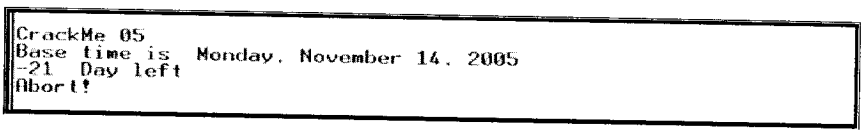


Рисунок 7.11 – Вигляд вікна після закінчення 20 днів

Нова інсталяція (тобто видалення і відновлення з оригіналу) заново не допомагає. Де записаний момент першого запуску? Можливо, у реєстрі? Це припущення неважко перевірити будь-яким монітором реєстру.

Перехоплення звернення до реєстру

Нагадаємо, що перегляд і редагування реєстру здійснюється програмою *regedit*.

Запустимо, наприклад, RegMon. Тепер всі звернення до реєстру протоколюватимуться, якщо тільки дана програма звертатиметься до реєстру. Запустивши програму Crack05.exe, ми бачимо, що наше припущення про те, що програма використовує реєстр, виправдалося. Прослідкуємо, як виглядає протокол (та частина інформації, що нас цікавить, знаходиться у самому кінці протоколу) при першому запуску захищеної програми (рис. 7.12) і при наступних запусках програми (рис. 7.13).

```
40 Crack05 OpenKey      HKCU\SOFTWARE\CRACK05   NOTFOUMD
41 Crack05 CreateKey    HKCU\SOFTWARE\CRACK05   SUCCESS hKey: 0xC29AF430
42 Crack05 SetValueEn   HKCU\SOFTWARE\CRACK05   SUCCESS 0x36D3A94F
43 Crack05 CloseKey     HKCU\SOFTWARE\CRACK05   SUCCESS
```

Рисунок 7.12 – Протокол програми RegMon при першому запуску Crack05.exe

```
35 Crack05 OpenKey      HKCr\SOrTWARE\CBACK05   SUCCESS hKey; 0xC29AFE60
36 Crack05 QueryValueEx HKCU\SOFTWARE\CRACK05   SUCCESS 0x36D3FC04
37 Crack05 CloseKey     HKCU\SOFTWARE\CRACK05   SUCCESS
```

Рисунок 7.13 – Протокол програми RegMon при наступних запусках Crack05.exe

Звичайно, можна видалити з реєстру розділ HKEY_CURRENT_USER\ SOFTWARE\CRACK05. Наступний запуск захист сприйме як перший. На процедуру розкриття пішло менше двох хвилин. Проте періодичне редагування реєстру просто незручне. Повноцінний злом припускає повне блокування захисного механізму, що ми і спробуємо зараз зробити.

Протокол дозволяє зрозуміти алгоритм роботи захисту. Спочатку програма намагається знайти в реєстрі розділ HKEY_CURRENT_USER\ SOFTWARE\CRACK05. Якщо він відсутній, то захист вважає, що на цьому комп'ютері запущена вперше і записує поточну дату. Інакше обчислюється кількість днів з моменту першого запуску. Можна змінити код так, щоб незалежно від результатів пошуку управління завжди передавалося на гілку першого запуску.

Використання дизасемблера IDA

Скористаємося дизасемблером IDA. Дизасемблюємо код програми Crack05 і, звернувшись до пункту меню *Search*, знайдемо в програмі фрагмент, який містить рядок `RegCreateKeyExA`. Посилань на цей рядок може бути декілька. Але в даному випадку ми, мабуть, знайшли те, що шукали: тут же поряд і знайомий нам рядок

```
aSoftwareCrack0 db 'SOFTWARE\CRACK05',
```

і декілька умовних переходів. Спробуємо розібратися (рис. 7.14).

Звернемо увагу на рядок `0x04010C2`. Не поспішатимемо змінювати цей умовний перехід.

Заглянувши в SDK, можна дізнатись, що функція *RegCreateKeyExA*() повертає ненульове значення у разі фатальної помилки. Значить, цей перехід міняти не можна. Результат завершення операції передається через локальну змінну `[esp+0x4]`. Якщо розділ був успішно створений, то функція повертає одиницю, інакше розділ вже існує. Тепер стає зрозуміло значення наступного фрагмента:

```
004010C8      cmp   dword ptr [esp+4],1
004010CD      jnz   short loc_401116
```

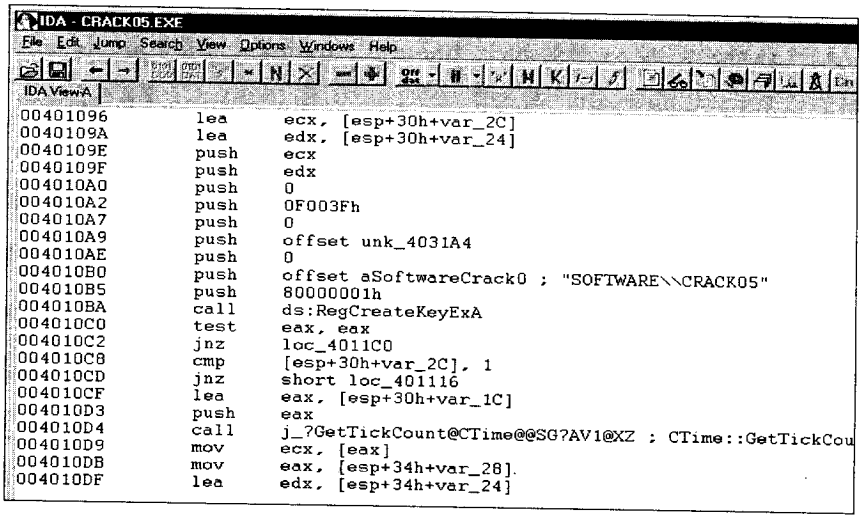


Рисунок 7.14 – Вікно дизасемблера IDA із знайденим фрагментом

Мабуть, саме цей перехід нам і потрібно поміняти. Встановлюємо курсор на команду, що цікавить нас. Запам'ятовуємо або записуємо значення відносної і абсолютної адреси цієї команди (їх можна побачити в самій нижній панелі дизасемблера). Редагувати код програми зручно за допомогою шістнадцяткових редакторів.

Використання шістнадцяткового редактора Hiew

Використовуємо для редагування тексту програми редактора Hiew. З командами цього редактора працювати досить просто і додаткових інструкцій для цього не потрібно. Меню в нижній частині вікна пропонує підказки в зручному вигляді. Відкриваємо файл, який треба відредагувати (рис. 7.15), і натискаючи клавішу *Enter*, слідкуємо, щоб код програми з'явився перед нами у вигляді команд асемблера (хоча при певному досвіді це можна зробити і в шістнадцятковому вигляді).

Знаходимо те місце, де потрібно внести зміни. Для цього за допомогою функціональної клавіші *F5* (команда *GoTo*) переходимо на потрібну нам команду: вводимо значення зсуву, яке ми запам'ятали раніше. Причому, якщо перед значенням адреси вказати крапку, то потрібно вводити абсолютний зсув, без крапки – відносний. Отже, ми перейшли на команду, що цікавить нас (рис. 7.16).

Тепер прийшов час вносити зміни в програму. Робити це потрібно після глибокого аналізу, щоб не зіпсувати програму безповоротно. Краще для цієї мети зробити копію і тренуватися саме на ній. Для внесення змін у код програми переходимо в режим редагування: клавіша *F3* (команда *Edit*). Корегувати можна двома способами:

– вносити зміни в шістнадцятковий код безпосередньо (рис. 7.17, а);
– використати клавішу *F2* (команда *Asm*) і ввести нову команду асемблера (рис. 7.17, б).

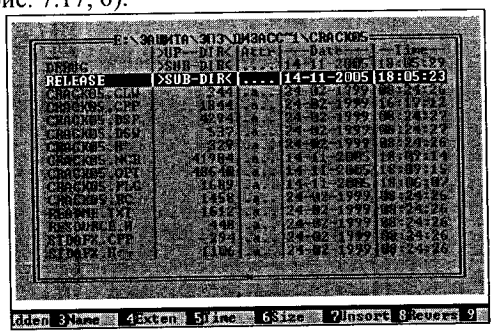


Рисунок 7.15 – Вікно редактора HIEW для пошуку файлу

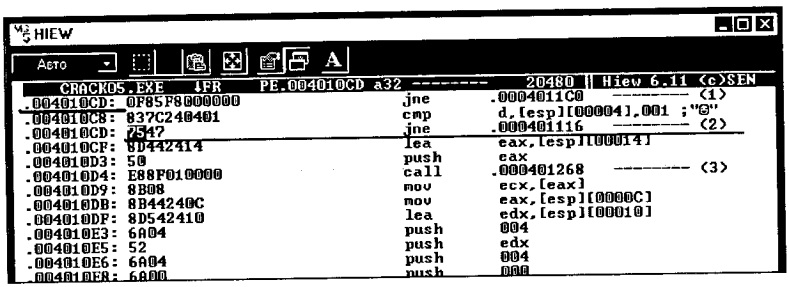
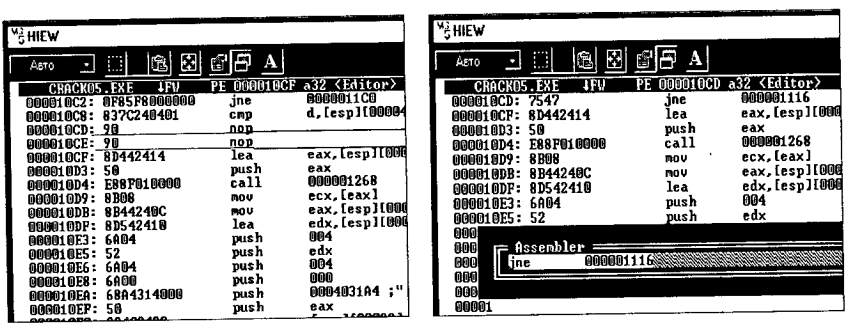


Рисунок 7.16 – Вікно редактора HIEW із знайденим фрагментом програми



а б

Рисунок 7.17 – Вікно редагування коду програми: а – режим зміни шістнадцяткового коду команди; б – режим зміни команди асемблера

Після того, як впевнимося, що команда виправлена правильно, можна запам'ятати наші зміни – клавіша *F9* (команда *UpDate*).

Виходимо з редактора (*F10*) і пробуємо запустити виправлену програму на виконання. Якщо ми все зробили правильно, програма повинна запускатися і не залежати від часу установлення.

Також в редакторі Hiew цікавою є інформація, яку видає нам команда *Header* (клавіша *F8*) – DOS-заголовок (рис. 7.18) і PE-заголовок (рис. 7.19).

Вище ми використовували для дослідження захищеної програми дизасемблер IDA. Проте, для цього можна використовувати і будь-який інший дизасемблер.

Висновок. Якщо для захисту програми використовується прив'язка до системного часу, то треба мати на увазі, що більшість подібних захистів не викликає складностей для їх злому. Можна за допомогою програм-моніторів прослідкувати, куди записується значення базового часу і внести зміни у певні файли операційних систем, а за допомогою дизасемблерів та шістнадцяткових редакторів можна побачити, які функції імпортуються і обійти елементи захистів.

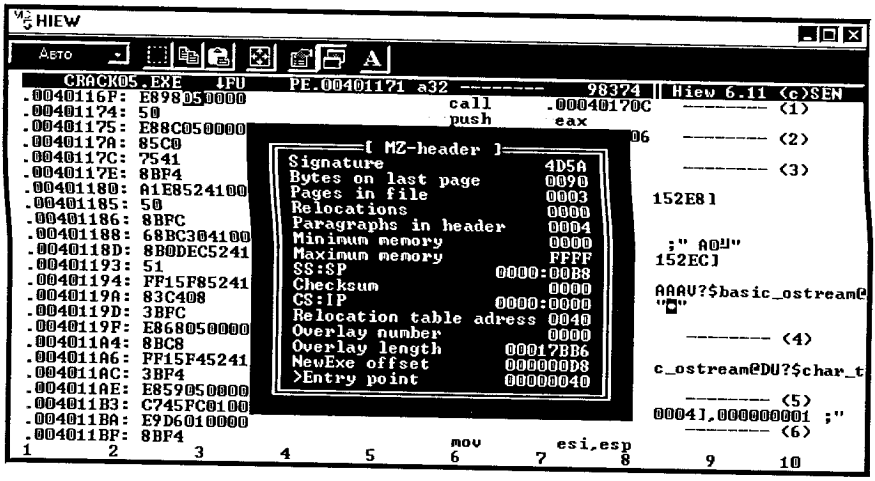


Рисунок 7.18 – Вікно параметрів DOS-заголовка виконуваного файлу

Чи можна якось удосконалити такі захисти? Так, завжди можна придумати якийсь карколомний прийом. Але для будь-якої головоломки можна знайти розв'язок – це тільки питання часу і кваліфікації зламника.

Існує декілька захисних механізмів, які не так очевидні, як вищеописані. Ось деякі з них.

Можна зберігати місяць першого запуску у полі десяткових долей секунди часу створення command.com. Хоча це не дуже підвищує стійкість захисту, але від некваліфікованих користувачів, які використовують у своїй роботі дискові сканери, захищає надійно.

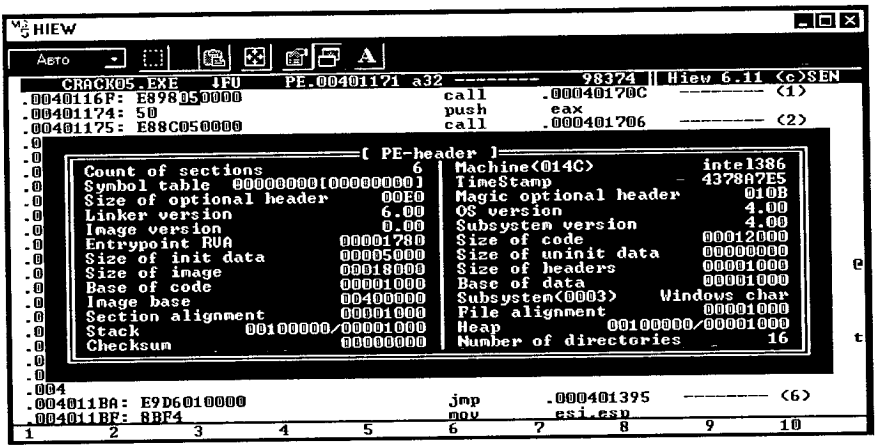


Рисунок 7.19 – Вікно параметрів PE-заголовка виконуваного файлу

Деякі захисти активно використовують для цієї мети незадіяні поля CMOS. Це дуже примітивний спосіб, що має ряд серйозних обмежень. Захист дуже помітний і легко перехоплюється. Дійсно, достатньо перехопити запис в порт 0x70, щоб знайти захист. Проте операційна система (на зразок Windows NT) не дозволить напряму звертатися до портів непривілейованим користувачам. Крім того, CMOS не видно по мережі. Нарешті, зарезервовані поля можуть бути використані в нових версіях, що призведе до конфліктів і, можливо, до серйозних наслідків.

Іноді зустрічаються досить оригінальні захисти, що не опитують системний час, а сканують диск у пошуках останнього створеного файлу, дату якого і приймають за поточну. Це надійно захищає від «переведення стрілок назад», проте вкрай ненадійно як метод. Дуже часто попадаються файли з неправильним часом створення. Вони можуть призвести до помилкового спрацьовування захисту, що ніяк не викличе захоплення у користувача. З другого боку, перехопити читання дати створення (останньої модифікації) файлу не складніше, ніж перехопити опитування системного часу. Механізми обох атак абсолютно ідентичні.

Захист обмеженням кількості запусків

Обмеження кількості запусків має багато спільного із захистом за часом з моменту першого запуску. Проте тепер замість початкового часу необхідно десь зберегти лічильник, що інкрементується (декрементується) під час кожного запуску програми.

Це спрощує аналіз протоколів монітора реєстру (або файлів). Дійсно, наведені вище приклади створювали тільки один розділ реєстру або один файл. Як правило, додаток середньої складності створює їх принаймні десятки, а то і сотні. Як знайти, яке з них має безпосереднє відношення до

захисного механізму? Універсальних порад в цій ситуації бути не може, і кожен випадок являє собою окрему головоломку.

Постійна зміна лічильника дозволяє, порівнявши протоколи різних запусків, знайти відмінності, яких звичайно буває небагато. Один з них і буде шуканим лічильником.

Приклад 3.

Розглянемо дизасемлювання програм, захищених обмеженням кількістю запусків. Продемонструємо на прикладі **crack09**, що захист може використовувати дуже складний і неочевидний формат. Знайти пару створених ним лічильників буде неважко. Але формат подання даних для нас буде загадкою. Здається, що обидва лічильники міняються довільним чином, то збільшуючись, то зменшуючись при кожній ітерації. Виникає навіть сумнів: а чи лічильники це взагалі? Можливо, це якісь інші службові дані?

З'ясувати істину нам допоможе налагоджувач або дизасемблер. Ми зробимо так. Спочатку переглянемо сегмент імпортованих функцій *.idata*. Бачимо, що програма звертається до реєстру (рис. 7.20).

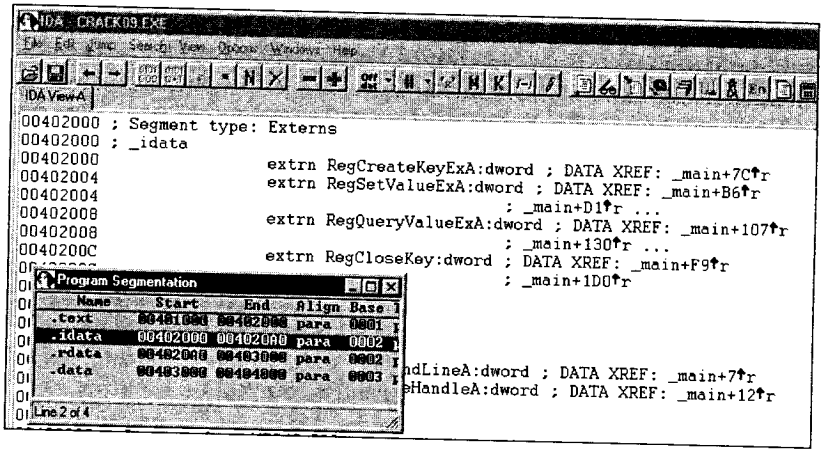


Рисунок 7.20 – Вигляд вікна сегмента імпортованих функцій в IDA

Шукаємо у сегменті *.text* рядок `RegCreateKeyExA`. З вищенаведеного знаємо, що перший перехід змінювати не потрібно (тут просто перевіряється правильність доступу до реєстру). Перейдемо на мітку, вказану у наступному переході. І справді, тут ми бачимо використання функції *RegQueryKeyExA()*. А поруч з викликом цієї функції бачимо посилання на декілька змінних *Count (Count, Count1, Count2)*. Звернемось до сегмента даних. Справді тут цих змінних декілька (рис. 7.21).

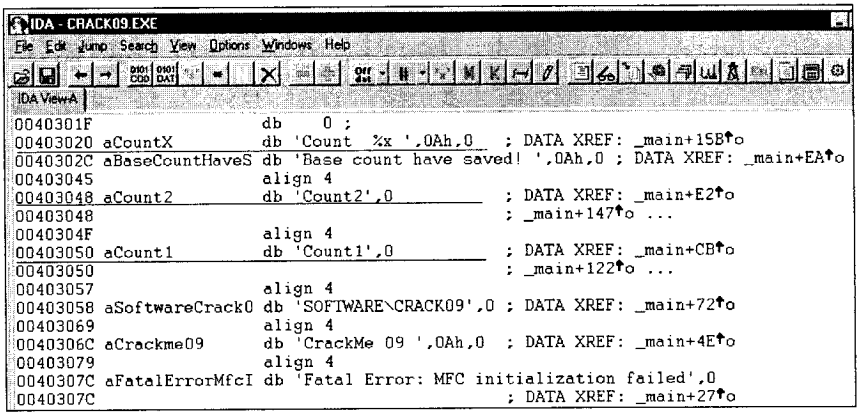


Рисунок 7.21 – Вигляд вікна сегмента даних у дизасемблері IDA

То яка ж з цих змінних справді лічильник? Аналізуємо код програми далі. Перехресні посилання допоможуть нам з'ясувати, який код читає або встановлює значення цього розділу реєстру. Не будемо наводити тут весь процес аналізу цілком, відзначимо тільки ключовий фрагмент:

```
.text:0040120F   mov     eax, [esp+5Ch+var_54]     ;Count2
.text:00401213   mov     edx, [esp+5Ch+var_4C]     ;Count1
.text:00401217 xor   eax, edx
```

Розшифровуємо значення лічильників. *Count1* насправді ключ, а *Count2* – зашифрований лічильник. Такий прийом дозволив більш-менш надійно приховати захисний механізм від некваліфікованого користувача, озброєного редактором реєстру. А далі бачимо:

```
.text '.00401219                dec     eax
```

Отже, тут зменшується значення лічильника на одиницю.

```
.text:00401220   test     eax, eax
.text:0040122F   jz       short loc_0_401296
```

Як неважко здогадатися, це і є той самий умовний перехід, який після закінчення відведених запусків додатка припиняє його роботу. Як тренування, рекомендується самостійно модифікувати його так, щоб програма працювала вічно. Зрозуміло, можна зробити інакше й видалити інструкцію dec – кому як подобається.

Аналізуємо код програми далі. Що ж записується у реєстр? А далі ми бачимо такий фрагмент:

```
.text:00401231     push   0
.text:00401233     call   ds:time
.text:00401239     push   eax
.text:0040123A     call   ds:sran
.text:00401240     add    esp, B
.text:00401243     call   ds:rand
```

Тут генерується випадкове число, міняється ключ шифрування після кожного запуску.

```
.text:00401249  mov       edi, [esp+5Ch+var_54]; Key
.text:0040124D  mov   ecx, [esp+5Ch+var_5] ; Real Count
.text:0040125B  xor       edi, eax
```

Далі зашифровується нове значення лічильника. І тепер це нове значення функції *RegSetValue()* заноситься у реєстр. Для більшої ясності наведемо фрагмент початкового тексту, що ілюструє вищевикладене:

```
Res=4;
RegQueryValueEx(hKey,»Counti»,0,&TYPE,(LPBYTE)sCountI,&res);
RegQueryValueEx(hKey,»Count2»,0,&TYPE,(LPBYTE)&Count2,&res);
Count2 = Count2^Count1;
Count2--;
printf («Count %x \n», Count 2);
if (!Count2) return 0;
srand(unsigned) time (NULL ) );
Count1 = (unsigned) rand I );
Count2 = Count2 ^ Count1;
RegSetValueEx(hKey,»Count1»,0,REG_DWORD,(CONST BYTE*) &Count1,4);
RegSetVaLueEx(hKey,»Count2»,0,REG_DWORD,(CONST BYTE *) &Count2,4);
```

Проте, більшість програмістів достатньо ледачі і зайняті, щоб активно використовувати подібні прийоми. Найчастіше лічильники записані в реєстрі «AS IS» і легко можуть бути змінені редактором реєстру на будь-яке інше значення, обмежене совістю зламника.

Отже, захисти з обмеженим часом або кількістю запусків схожі між собою. Вони досить прості та не викликають великих труднощів при зломі. Рекомендації при реалізації цього захисту такі самі, як і у попередньому випадку.

Контрольні запитання

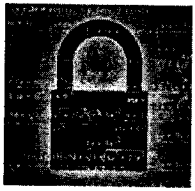
1. Які програмні засоби необхідні для здійснення статичного дослідження виконуваних модулів?
2. В чому полягає захист програм паролем? Де може бути збережений пароль?
3. Як здійснюють захист за допомогою обмеження часу використання? Для якої категорії програм цей захист доцільно використовувати?
4. Де можна зберігати час використання програм? Яким чином можна здійснювати злом такого захисту?
5. В чому полягає захист обмеженням кількості запусків? Як можна приховувати змінні, які містять кількість запусків? Де її можна зберігати?

Порядок виконання лабораторної роботи

1. Дослідити інтерфейс і можливості програм для дослідження і модифікації виконуваних кодів: IDAPro та HIEW.
2. Користуючись вищенаведеним описом, навчитись дизасемблювати запропоновані програми (директорія «Приклади»).
3. Запустити на виконання програму відповідно до індивідуального завдання (папка «Завдання»).
4. Дослідити, який метод захисту використано у даній програмі (для цього можна скористатися програмами RegMon, FileMon.
5. Дизасемблювати код програми і відредагувати його таким чином, щоб зняти захист (засоби для дизасемблювання та редагування – у директорії «Instruments»).
6. Зробити висновки щодо ефективності даного виду захисту: причини недостатньої стійкості та рекомендації щодо його покращення.
7. *Згідно з номером індивідуального завдання «зламати» програму з директорії «Додатково!!!\CrackMe», користуючись при цьому наданою документацією (файл «Приклади практичної реалізації злому»).

   У звіті відобразити:
   - процес роботи захищеної програми;
   - процесс дослідження і зняття захисту;
   - процес роботи «зламаної» програми.

Лабораторна робота № 8



РОЗРОБКА І РЕАЛІЗАЦІЯ ВБУДОВАНОГО ЗАХИСТУ ПРОГРАМ

Мета і задачі

- Ознайомитись з основними групами методів для захисту програм від дизасемблювання.
- Дослідити основні методи обфускації програмного коду. Знати види і рівні обфускації коду.
- Навчитись на практиці використовувати у програмах основні методи обфускації і заплутування коду.
- Вміти досліджувати коди розроблених програм до і після захисту з метою доведення його ефективності.
- Навчитись обґрунтовувати вибір програмних засобів для реалізації систем захисту програм.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Основні методи протидії дизасемблюванню програм

Виділимо декілька найзагальніших підходів до захисту програмного забезпечення від дизасемблювання.

1. *Шифрування коду* – один з найпоширеніших та надійних методів захисту від статичного дослідження.
2. *Маніпулювання заголовками EXE-файлів* – метод, побудований на використанні властивостей структури виконуваних файлів і розробці навісного захисту.
3. *Обман дизасемблера.* Методи цієї групи полягають у тому, щоб заплутати дизасемблер: підсунути дані замість коду, дезорієнтувавши його логіку, повести його по помилковому сліду, підсунути зайві фрагменти коду і т. д. Всі ці способи обману можна поділити на такі групи:
   - різноманітні методи обфускації коду;
   - ускладнення логіки програм;
   - різноманітні додаткові методи боротьби з інтерактивними та автоматичними дизасемблерами.

   Це авангардні і досить перспективні методи захисту ПЗ не тільки від дизасемблювання, а й від налагоджування.
4. Методи *емуляції*, серед яких виділяють:
   - емуляцію процесора;
   - емуляцію мультизадачності.

Захист програм шляхом обфускації

Обфускація (*«obfuscation»*) – це один з методів захисту програмного засобу, який дозволяє ускладнити процес реверсивної інженерії початкового коду програми, що захищається. Суть процесу обфускації полягає в тому, щоб заплутати програмний код і усунути більшість логічних зв'язків у ньому, тобто трансформувати його так, щоб він був дуже важкий для вивчення і модифікації сторонніми особами (будь то зловмисники, або програмісти які збираються дізнатися унікальний алгоритм роботи програми, що захищається). Обфускація може застосовуватися не тільки для захисту програмних продуктів, вона має ширше застосування (наприклад, може бути використана творцями вірусів для захисту їх творінь і т. д.).

Обфускація буває декількох видів:

- лексична обфускація (символьна обфускація);
- обфускація даних;
- обфускація графа потоку керування.

Лексична обфускація

Лексична обфускація полягає в форматуванні коду програми, зміні його структури таким чином, щоб він став нечитабельним, менш інформативним і важчим для дослідження дизасемблерами і декомпіляторами.

Обфускація такого виду включає в себе певні складові.

1. *Видалення необов'язкових конст рукцій* мови програмування:
   – видалення усіх коментарів в коді або зміна їх на дезінформуючі;
   – видалення пробілів, відступів і т. д.
2. *Символьна обфускація* – заміна імен ідентифікаторів (змінних, масивів, структур, функцій і т. д.) на набори символів, які важко сприймати людині (символьна обфускація найбільш легко реалізується):
   – перейменування методів, змінних і т. д. в набір безглуздих символів. Наприклад, був метод класу GetPassword(), після обфускації даний метод матиме ім'я KJHS92DSLKaf();
   – перейменування найменувань змінних і методів у коротші. Проходячи по всіх класах, методах, вони замінюють імена на їх порядкові номери. Наприклад, був метод GetConnectionString(), а став _0();
   – використання ключових слів мови програмування;
   – використання імен, що міняють сенс. Тут швидше діє психологічний чинник. Припустимо, клас з назвою SecurityInformation з методом GetInformation() став класом Car з методом Wash().

Обфускація даних

До складніших обфускаторів, які дають більше гарантій того, що наш код не зможуть зрозуміти зловмисники, належать обфускатори другого порядку, робота яких пов'язана із трансформацією структур даних. Вона вважається більш складною, більш сучасною і використовуваною.

Наведемо деякі способи здійснення обфускації даних.

1. *Розділення змінних*. Змінні фіксованого діапазону можуть бути розділені на дві й більше змінних. Для цього змінну V, що має тип x розділяють на k змінних $v_1,...,v_k$ типу y, тобто $V=v_1,...,v_k$. Потім створюється набір операцій, що дозволяють витягати змінну типу x зі змінних типу y і записувати змінну типу x у змінні типу y. Як приклад розділення змінних можна розглянути спосіб подання однієї змінної B логічного типу (*boolean*) двома змінними b_1 і b_2 типу короткого цілого (*short*), значення яких буде інтерпретуватися таким чином:

B	b_1	b_2
false	0	0
true	0	1
true	1	0
false	1	1

Тоді фрагмент коду (мова C++):

```
bool B ;
B = false ;
if (B) {  ...  }
```

буде перетворено у такий:

```
short b1, b2 ;
b1 = b2 = 1 ;     // або b1 = b2 = 0
if (!(b1 & b2)) {  ...  }
```

2. *Заміна подання* (або кодування). Наприклад, цілочисельну змінну i у нижченаведеному фрагменті коду (мова C):

```
...
int  i=1;
...
while (i<1000)
{   ... A[i] ...
    i++ ;
}
```

можна замінити, виразом $i=c_1 \cdot i+c_2$, де c_1, c_2 – константи. В результаті наведений код зміниться й буде складніший для сприйняття:

```
...
int  i=11;
int  c1=8, c2=3 ;
...
while (i<8003)
{ ... A[(i-3)/8] ...
  i+=8 ;
}
```

3. *Реструктурування масиву – розділення* одного масиву на декілька підмасивів. Наприклад, масив A можна розділити на підмаси $A1$ і $A2$, де масив $A1$ міститиме парні, а $A2$ – непарні позиції елементів масиву A. Тому фрагмент

```
int  A[] = {a, b, c, d, e, f} ;
i = 3 ;
A[i] = ... ;
```

можна замінити на такий:

```
int A1 = {2, 4, 0} ;
int A2 = {1, 3, 5} ;
i = 3 ;
```

```
if (($i % 2) == 0) { $A1[$i / 2] = ... ; }
else { $A2[$i / 2] = ... ; }
```

4. *Реструктурування масиву – згортання.* Наприклад, одновимірний масив A розмірністю 6, можна замінити двовимірним масивом B розмірністю 2×3, після чого код

```
int  A[] = {1, 2, 3, 4, 5, 6} ;
for (int  i = 0 ; i < 6 ; i++)
{       A[i] = A[i] + 1 ;
        print f(«%d\n», A[i]) ;
}
```

можна змінити на такий:

```
int  A[2][3] = { {1,2,3}, {4,5,6} };
for (int  i = 0 ; i < 2 ; i++)
{       for (int  ii = 0 ; ii < 3 ; ii++)
        {       A[i][ii] = A[i][ii] + 1 ;
                print f(«%d\n», A[i][ii]) ;
        }
}
```

5. *Розгортання циклів* (loop unrolling). Сутність: тіло циклу розмножується два або більше разів, а умова виходу з циклу і оператор збільшення лічильника відповідним чином модифікуються. Наприклад,

до модифікації

```
for (i=1; i=<n; i++)
{
        // тіло циклу
}
```

після модифікації

```
for (i=1; i<n-1; i++)
{    // тіло циклу  }
// тіло циклу
```

6. *Розкладання циклів (loop fission).* Сутність: цикл зі складним тілом розбивається на декілька окремих циклів з простими тілами й з тим самим простором ітерування.

до модифікації

```
for (i=1; i<n; i++)
{  a[i]+= c ;
   x[i+i]=d+x[i+1]*a[i];
}
```

після модифікації

```
for (i=1; i<n; i++)
     {       a[i] += c;       }
for (i=1; i<n; i++)
{    x[i+i]=d+x[i+1]*a[i];   }
```

7. *Пониження розмірності індексного простору.* Сутність: перехід від багаторазового вкладеного циклу до єдиного циклу за допомогою певних формул.

до модифікації

```
enum {N=128};
typedef  double MATR [N][N];

void mm (MATR m1,MATR m2,MATR r)
{ int i,j,k;
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        r[i][j]=0;
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      for (k=0; k<N; k++)
        r[i][j]+=m1[i][k]*m2[k][j];
      . . . .       }
```

після модифікації

```
enum {N=128};
typedef double MATR[N][N];

void mm (MATR m1,MATR m2, MATR r)
{  int i,j,k;
   for (i=0; i<N*N; i++)
       r[i/N][i%N]=0;
   for (i=0; i<N*N*N; i++)
      r[i/(N*N)][i%(N*N)/N]+=
            m1[i/(N*N)][i%N]*
            m2[i%N][i%(N*N)/N];
       . . . .
  }
```

8. *Використання алгебраїчних перетворень* для формування неістотного коду у тілі програми. Наприклад,

```
sin²x+cos²x = 1;    cos π = -1;    ln e = 1; …
```

9. *Використання непрозорих предикатів у вигляді комбінаторних тотожностей.* Наприклад,

$$2^n = \sum_{k=0}^{n} C_n^k \qquad 0 = \sum_{k=0}^{n} (-1)^k C_n^k \qquad n = \sum_{k=0}^{n} (-1)^k 4^{n-k} C_{2n-k+1}^k - 1$$

Способи реалізації ускладнення логіки

Як один з методів захисту від зворотної інженерії застосовується маскування програм. Кажучи неформально, *маскування програми* – це таке перетворення її тексту, яке повністю зберігає функціональність, але робить розуміння, зворотну інженерію і модифікацію тексту програми завданням неприйнятно високої вартості. Можна сказати, що маскування програм – це обфускація графа потоку керування програмою.

Об'єктами маскування є тексти реальних програм, що складаються з сотень функцій по декілька сотень рядків кожна. Замасковані програми повинні вкладатися в обмеження обчислювальної системи, що не може не відбитися на використовуваних методах маскування.

Існує декілька груп методів ускладнення логіки, які можна комбінувати з іншими методами.

1. Маніпулювання функціями

- *Відкрита вставка функцій* (*function inlining*) полягає в тому, що тіло функції підставляється в місце її виклику.
- *Винесення групи операторів* (*function outlining*). Дане перетворення є оберненим до попереднього і добре доповнює його. Деяка група операторів вихідної програми виділяється в окрему функцію. За необхідності створюються формальні параметри.
- *Усунення бібліотечних викликів* (*eliminating library calls*). Більшість програм мовою Java широко використовують стандартні бібліотеки. Оскільки семантика бібліотечних функцій добре відома, такі виклики можуть дати корисну інформацію при зворотній інженерії програм. Таке перетворення не змінить істотно час виконання програми, але збільшить її розмір.
- *Переплетення функції* (*function interleaving*). Ідея цього перетворення в тому, що дві або більше функцій поєднуються в одну. Списки параметрів вихідних функцій поєднуються і до них додається ще один параметр, який дозволяє визначити, яка функція в дійсності виконується. Заплутана функція замість операторів if, for і т. д. буде містити оператор switch, розташований всередині нескінченного циклу.
- *Клонування функцій* (*function cloning*). Можна створити декілька клонів і до кожного з них застосувати різний набір заплутуваних перетворень.

2. *Використання непрозорих предикатів*

Основною проблемою при проектуванні перетворень, що заплутують граф потоку керування, є те, як зробити їх не тільки дешевими, але й стійкими. Для забезпечення стійкості більшість перетворень ґрунтується на введенні *непрозорих змінних* і *непрозорих предикатів* (*opaque predicates*). Сила таких перетворень залежить від складності аналізу непрозорих предикатів і змінних. Аналогічно, предикат **P** називається **непрозорим**, якщо його значення відоме в момент заплутування програми, але важко відновлюване після його завершення.

Використання непрозорих предикатів можна реалізувати по-різному.

- *Різні способи доступу до елементів масиву*. Нехай у програмі створено масив *a*, що ініціалізується заздалегідь відомими значеннями, а далі в програму додаються змінні *i* та *j*, в яких зберігатимуться індекси елементів цього масиву. Тепер непрозорі предикати можуть мати вигляд: *a[i]==a[j]*. Якщо до того ж змінні *i* та *j* у програмі змінюються, то існуючі методи статичного аналізу дозволять тільки визначити, що і *i*, і *j* вказують на будь-який елемент масиву *a*. Як простий спосіб можна використати розміщення всіх локальних змінних одного типу в масиві.
- *Використання покажчиків на спеціально створювані динамічні структури*. У цьому підході в програму додаються операції по створенню структур даних (списків, дерев), і додаються операції над покажчиками на ці структури, підібрані таким чином, щоб зберігалися деякі інваріанти, які й використовуються як непрозорі предикати. Один з простих способів: всі значення змінних усіх типів зберігаються в списку, який розміщується в динамічній пам'яті.
- *Побудова складних булевих виразів* за допомогою еквівалентних перетворень із формули *true*. У найпростішому випадку ми можемо взяти *k* довільних булевих змінних $x_1 \ldots x_k$ і за допомогою еквівалентних алгебраїчних перетворень, коли частина дужок або всі дужки розкриваються, побудувати з них тотожності, наприклад:

$$1=(x_1 \cup x_1) \cap \ldots \cap (x_k \cup x_k);\ 0=(x_1 \cup \bar{x}_1) \cap \ldots \cap (x_k \cup \bar{x}_k).$$

Це можна використати у багатьох випадках. Так, фрагмент коду:

```
i = 1 ;
while (i<101)
{     ...
      i++ ;
}
```

після розширення умов циклу матиме вигляд:

```
i = 1 ;
j = 100 ;
while ((i<101) && (j*j*(j+1)*(j+1)%4 == 0)(t))
 {       ...
         i++ ;
         j = j*i+3 ;
 }
```

– *Використання комбінаторних тотожностей.* Воно може бути використано для ускладнення логіки програми всюди: і при вбудовуванні функцій, і при клонуванні функцій, і при розгортанні циклів, і при вбудовуванні мертвого та недосяжного кодів тощо.

3. *Внесення недосяжного, мертвого або надлишкового коду*

Якщо в програму внесені непрозорі предикати видів $\mathbf{P^F}$ або $\mathbf{P^T}$, гілки умови, які відповідають умові *true* у першому випадку й умові *false* у другому випадку, ніколи не будуть виконуватися.

– *Недосяжний код* (*unreachable code*) – це фрагмент програми, що ніколи не виконується. Ці гілки можуть бути заповнені довільними обчисленнями, які схожі на дійсно виконуваний код. Оскільки недосяжний код ніколи не виконується, дане перетворення впливає тільки на розмір заплутаної програми, але не на швидкість її виконання.

– *Мертвий код* (*dead code*), на відміну від недосяжного, у програмі виконується, але його виконання ніяк не впливає на результат роботи програми. При внесенні мертвого коду розробник повинен бути впевнений, що вставляє фрагмент, який не може впливати на код, який обчислює значення функції.

– *Надлишковий код* (*redundant code*), на відміну від мертвого коду, виконується, і результат його виконання використовується надалі в програмі, але такий код можна спростити або зовсім видалити, оскільки обчислюється або константне значення, або значення, уже обчислене раніше. Для внесення надлишкового коду можна використати алгебраїчні перетворення, ввести в програму математичні тотожності.

Контрольні запитання

1. Які групи методів використовують для захисту від несанкціонованого дослідження коду програм? Дайте загальну характеристику цим методам.
2. Що таке обфускація і які рівні обфускації існують?
3. Як здійснюється лексична обфускація? З яких етапів вона складається?
4. Що означає обфускація даних? Які види обфускації даних існують?
5. Наведіть приклади обфускації даних.
6. В чому сутність маскування програм і яка його мета?
7. Які заплутувані перетворення функцій можна здійснити для реалізації захисту програм від дослідження?
8. Що таке непрозорі предикати? Де використовуються непрозорі змінні і непрозорі предикати?
9. Де і як у програмах можна використати комбінаторні тотожності?
10. Які види внесення неістотного коду ви знаєте? Що таке недосяжний, мертвий та надлишковий коди?

Порядок виконання лабораторної роботи

1. Ознайомитись з основними методами обфускації.
2. Розробити програми згідно з індивідуальним завданням. При цьому:
   а) перша програма повинна здійснювати захист, не використовуючи обфускацію;
   б) друга програма повинна містити елементи обфускації свого коду.
3. Дизасемблювати виконуваний код першої програми і здійснити її злом.
4. Дизасемблювати виконуваний код другої програми і здійснити її злом.
5. Підготувати звіт щодо ефективності даного виду захисту і складності злому.

Варіанти індивідуальних завдань

	Спосіб захисту	Метод обфускації	Мета злому
1.	Захист за допомогою серійного ключа	Реструктурування масивів (згортання)	Отримати серійний ключ
2.	Захист паролем	Зміна подання	Отримати пароль
3.	Захист обмеженням кількості запусків	Використання непрозорих предикатів (комбінаторні тотожності)	Отримати кількість запусків
4.	Захист прив'язкою до терміну використання (до дати)	Використання складних булевих виразів	Обійти перевірку дати
5.	Захист автентифікацією (логін + пароль)	Реструктурування масиву (розділення)	Виділити пароль і (або) логін
6.	Захист за допомогою серійного ключа	Розкладання циклів	Отримати ключ
7.	Захист паролем	Шифрування (наприклад, зсув по таблиці ASCII-кодів)	Отримати пароль
8.	Захист обмеженням кількості запусків	Використання алгебраїчних перетворень	Змінити кількість запусків
9.	Захист прив'язкою до терміну використання (до дати)	Використання способу подання (наприклад, застосувати структури)	Змінити базову дату
10.	Захист автентифікацією (логін + пароль)	Пониження розмірності індексного простору	Виділити пароль і (або) логін

ГЛОСАРІЙ

Несанкціонований доступ (НСД) – unauthorized access – нелегальні дії щодо використання, зміни та знищення виконуваних модулів.

Система захисту від несанкціонованого доступу (system of protection against unauthorized access) – комплекс програмних засобів, що забезпечують ускладнення або заборону нелегального розповсюдження, використання і/або зміну програмних продуктів.

Надійність системи захисту (reliability of protection) – здатність протистояти спробам проникнення в алгоритм її роботи і обходу механізмів захисту.

Зломом програми (breaking program) – порушення функціональності об'єктів захисту програмного забезпечення.

Обфускація (obfuscation) – це один з методів захисту програмного коду, який дозволяє ускладнити процес реверсивної інженерії коду програмного продукту, що захищається.

Хук (Hook) – механізм перехоплення особливою функцією подій (таких, наприклад, як повідомлення від маніпулятора миші або клавіатури) до того, як вони дійдуть до програмного додатка.

Динамічне дешифрування – дешифруваня у міру виконання конфіденційної частини програми.

Relocation Table – таблиці для настроювання адрес. Складається зі значень у форматі <сегмент : зсув>. До зсувів у завантажувальному модулі, на яких указують значення в таблиці, після завантаження програми в пам'ять повинна бути додана сегментна адреса, з якої завантажена програма.

Таблиця об'єктів або таблиця розділів (object) – сукупність даних певного призначення: про експортовані та імпортовані функції, про ресурси, про переміщення (relocations) і т. д., які компактно розміщені у виконуваному файлі.

Х-код – частина коду програми, яку ми збираємося впроваджувати у програму з метою її захисту.

Лексична обфускація полягає в форматуванні коду програми, зміні його структури таким чином, щоб він став нечитабельним, менш інформативним і важчим для дослідження дизасемблерами і декомпіляторами.

Символьна обфускація – заміна імен ідентифікаторів (імен змінних, масивів, структур, функцій, процедур і т. д.) на самовільні довгі набори символів, які важко сприймати людині.

Обфускація даних – ускладнення розуміння даних програми, пов'язане із трансформацією структур даних.

Маскування програми – це таке перетворення її тексту, яке повністю зберігає функціональність, але робить розуміння, зворотну інженерію і модифікацію тексту програми завданням неприйнятно високої вартості.

Змінна є непрозорою, якщо існує деяка властивість щодо цієї змінної, яка відома в момент заплутування програми, але важко відновлюється після того, як заплутування завершене.

Непрозорі предикати (opaque predicates) – вираз, значення якого відоме в момент заплутування програми, але важко відновлюване після його завершення.

Недосяжний *код (unreachable code)* – фрагмент програми, що ніколи не виконується.

Мертвий *код (dead code)* – фрагмент коду, що у програмі виконується, але його виконання ніяк не впливає на результат роботи.

Надлишковий код (redundant code) – фрагмент коду, що виконується, і результат його виконання використовується надалі в програмі, але такий код можна спростити або зовсім видалити, оскільки обчислюється або константне значення, або значення, уже обчислене раніше.

Автоматичні дизасемблери – програмні засоби, що аналізують код виконуваного файлу й формують відповідний йому вихідний текст або лістинг.

Інтерактивні дизасемблери – програмні засоби, що формують вихідний текст/лістинг виконуваного коду програми так само, як це роблять автоматичні дизасемблери, однак відрізняються наявністю потужного користувацького інтерфейсу, що значно полегшує аналіз дизасемблерної програми.

Дамп пам'яті (memory dump) – це копія вмісту оперативної пам'яті, що знаходиться на жорсткому диску або іншому енергонезалежному пристрої пам'яті.

Дампери (dampers) – програми для знання дампу.

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Дудатьєв А. В. Захист програмного забезпечення. Ч. 1 : навчальний посібник / Дудатьєв А. В., Каплун В. А, Семеренко В. П. – Вінниця : ВНТУ, 2005. – 140 с.
2. Каплун В. А. Захист програмного забезпечення. Ч. 2 : навчальний посібник / Ю. В. Баришев, О. В. Дмитришин. – Вінниця : ВНТУ, 2013. – 151 с.
3. Казарин О. В. Теория и практика защиты программ / Казарин О. В. – М. : МГУЛ, 2004. – 450 с.
4. Соколов А. В. Защита от компьютерного терроризма : справочное пособие / А. В. Соколов, О. М. Степанюк. – СПб. : БХВ-Петербург, Арлит, 2002. – 496 с.
5. Щеглов А. Ю. Защита компьютерной информации от несанкционированного доступа : учебное пособие / Щеглов А. Ю. – СПб. : Наука и Техника, 2004. – 384 с.
6. Щербаков А. Ю. Защита от копирования: построение программных средств / Щербаков А. Ю. – М. : Эдель, 1992. – 80 с.
7. Румянцев П. В. Работа с файлами в Win 32 API / Румянцев П. В. – М. : Горячая линия-Телеком, 2002. – 216 с.
8. Румянцев П. В. Исследование программ Win32: до дизассемблера и отладчика / Румянцев П. В. – М. : Горячая линия-Телеком, 2004. – 367 с.
9. Абашев А. А. Ассемблер в задачах защиты информации / Абашев А. А., Жуков И. Ю., Иванов М. А. – М. : Кудиц-Образ, 2004. – 544 с.
10. Касперски К. Компьютерные вирусы изнутри и снаружи / Крис Касперски. – СПб. : Питер, 2006. – 527 с.
11. Касперски К. Техника и философия хакерских атак / Крис Касперски. – М. : Солон-Р, 1999. – 272 с.
12. Войтович О. П. Методичні вказівки до виконання курсового проекту з дисципліни «Захист програмного забезпечення» для студентів напряму підготовки 6.170101 «Безпека інформаційних і комунікаційних систем» / О. П. Войтович, В. А. Каплун. – Вінниця : ВНТУ, 2010. – 57 с.
13. Чернов А. В. Интегрированная среда для исследования «обфускации» программ. Доклад на конференции, посвящённой 90-летию со дня рождения А. А. Ляпунова. Россия, Новосибирск, 8–11 октября 2001 года // Електронный ресурс: http://www.ict.nsc.ru/ws/Lyap2001.
14. Домашев А. В. Программирование алгоритмов защиты информации : учебное пособие / Домашев А. В., Попов В. О., Правиков Д. И. – М. : Нолидж, 2002. – 416 с.

Навчальне видання

Каплун Валентина Аполінаріївна
Дмитришин Олександр Васильович
Баришев Юрій Володимирович

ЗАХИСТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Лабораторний практикум

Редактор Є. Плетньова

Оригінал-макет підготовлено В. Каплун

Підписано до друку 3.05.2017 р.
Формат 29,7×42¼. Папір офсетний.
Гарнітура Times New Roman.
Ум. др. арк. 4,39.
Наклад 50 прим. Зам. № 2017-085.

Видавець та виготовлювач
Вінницький національний технічний університет,
інформаційний редакційно-видавничий центр.
ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95,
м. Вінниця, 21021.
Тел. (0432) 59-85-32, 59-87-38.
press.vntu.edu.ua.
Email: kivc.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.