РІВНЕНСЬКИЙ ДЕРЖАВНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ

Факультет математики та інформатики

Кафедра інформатики та прикладної математики

«До захисту допущено»
Завідувач кафедри
_____ Батишкіна Ю.В.
«___»_____20___ р.

Дипломний проект (робота)

ступеня «Магістр»

з спеціальності 113 – Прикладна математика

на тему: Розробка комплексу програм для захисту програмного забезпечення на основі web-ліцензування

Виконав: студент 2 курсу, групи М-ПМ-21
Бєліч Андрій Олександрович _____

Керівник: к.т.н., доц. Сінчук А. М. _____

Консультант: к.т.н., доц. Кот В.В. _____

Рецензент: к.т.н., доц. Назарук М.В. _____
к.т.н., доц. Бабич С.М. _____

Засвідчую, що у цьому дипломному проекті немає запозичених матеріалів з праць інших авторів без відповідних посилань.
Студент _____

Рівне – 2020 року

Зміст

СПИСОК УМОВНИХ ТЕРМІНІВ ТА ПОЗНАЧЕНЬ

x86 – процесорна архітектура розроблена компанією Intel, зазвичай мається наувазі все, що новіше чим i386 вітки IA-32.

x86_64 (x64) – 64-бітна архітектура основана на 32-бітній x86 і сумісна з нею, розроблялась компанією Advanced Micro Devices як альтернатива Intel Itanium, в якій була відсутня зворотня сумісність.

PIC (position independent code) – загальна назва коду що може виконуватись незалежно від розміщення в памяті, також використовується для коду зібраного з механізмами релокації (або аналогів).

Структура – композитний тип данних за допомогою якого описується структура пам’яті

SQL – мова структурованих запитів, що використовується для взаємодії з базами даних.

SQLITE – компактна вбудована СУБД з відкритим кодом.

PE (Portable Executable) – формат виконуваних файлів, об’єктного коду, динамічних бібліотек, що використовується в 32- та 64-розрядних версіх операційної системи Windows.

ВСТУП

Актуальність теми. Технічна сфера розробки програмного забезпечення (в подальшому - ПЗ) з кожним роком збільшує грошовий обіг. Вже більше 30 років використовують різного роду ПЗ для рішення багатьох банальних задач: від розробки календаря до комп'ютерних ігор. Ці, та багато інших розробок, стали основною частиною заробітку та дозвілля. Разом з цим, збільшилась і вартість розробки програмних продуктів, а, відповідно, розробникам потрібні інструменти для захисту при розповсюдженні ПЗ.

Для цієї задачі є декілька шляхів вирішення. Найзручніший метод, зі сторони розробника, – програмний комплекс що підтримується іншою командою та не впливає на основний робочий процес. Такий програмний комплекс дозволяє не витрачати ресурси на розробку надійної захисної системи.

Основною невирішеною проблемою подібних програмних комплексів – їх ціна та складність у використанні для розробників, а саме, при необхідності домогтись високого рівня захисту.

Збитки від піратства з кожним роком ростуть, наприклад, у 2019 році вони перевищували 9.1 мільярдів тільки у США [13], дані за 2020 нажаль не публікуються, але за прогнозами аналітиків в середньому сума зростає на 15% за рік. Основною причиною являється складність розробки, або ж доступ до надійних захисних систем.

Об'єкт. Технологічний процес розробки комплексу програм для захисту програмного забезпеченя.

Предмет. Розроблений програмний комплекс дозволяє розробникам програмного забезпечення, з прикладанням мінімальних зусиль, використовувати систему ліцензування та управляти ліцензіями на програмне забезпечення.

Метою роботи є: створення комплексу програм, що буде надавати систему (інструменти) ліцензування для розробників в максимально простій формі під ПЗ що розробляється на операційній системі сімейства Windows.

Практична цінність. Розроблений комплекс програмного забезпечення може бути використаний при подальших дослідженнях, а також може знайти своє застосування у сфері розробки ПЗ в якості захисних інструментів.

Завдання дослідження:

- зібрати, дослідити та проаналізувати теоретичну матеріали по темі дипломного проєкту, в особливості розглянути будову PE файлів;
- розглянути можливості реалізації захисних систем;
- дослідити існуючі рішення та, на основі цього, побудувати технічне завдання дипломного проєкту;
- реалізувати роботу розробленого комплексу програмного забезпечення.

Апробація результатів дослідження:

- Звітна науково-практична конференція професорсько-викладацького складу Рівненського державного гуманітарного університету, яка відбулася 14-15 травня 2020 року року. Рівне.
- IV Всеукраїнська науково-практична конференція здобувачів вищої освіти та молодих науковців «Прикладні аспекти інформаційного забезпечення та обґрунтування технічних і управлінських рішень» у Рівненському державному гуманітарному університеті, яка відбулася 20 травня 2020 року. Рівне. (Додаток А).

Розділ 1.
АНАЛІЗ ВХІДНИХ ДАНИХ ТА РОЗРОБКА ТЕХНІЧНОГО ЗАВДАННЯ

1.1. Будова PE-файлів

PE – це формат виконуваних файлів для операційних систем сімейства Windows на архітектурах x86 та x86_64 [1]. На даний момент є два варіанта формату: PE та PE+. Відповідно PE для архітектури x86, а PE+ для x86_64. Формат можна поділити на певні узагальнені частини: заголовки (саме в множині, оскільки заголовок не один), основні секції, інші данні. Загальна структура формату представлена на рисунку 1.

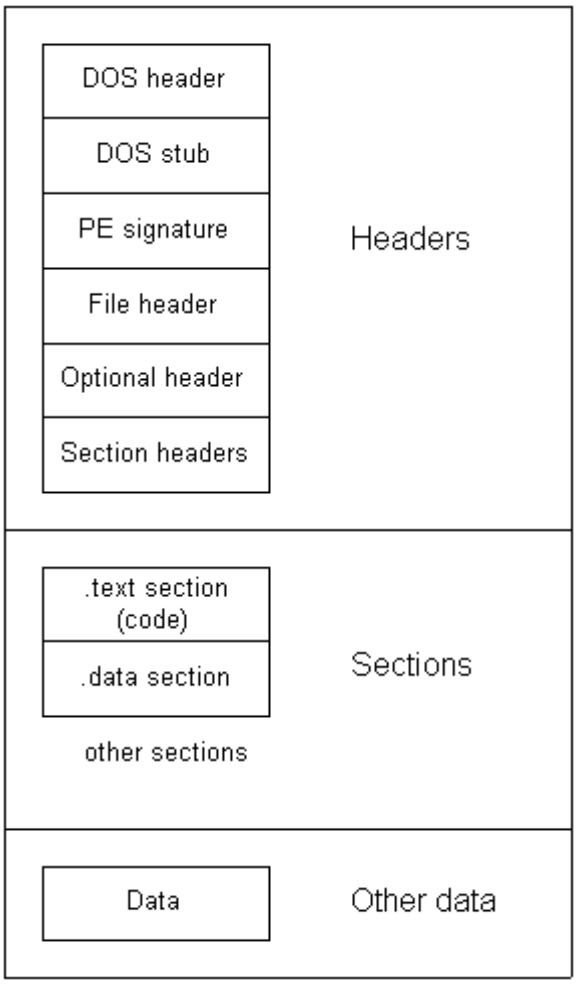

Рис. 1. Загальна структура виконуваного PE-файлу[6].

DOS заголовок (IMAGE_DOS_HEADER) – це перша в структура за якої розпочинається файл, являється в певній мірі пережитком минулого – часів операційної системи MS-DOS.

Структура IMAGE_DOS_HEADER:

```
typedef struct _IMAGE_DOS_HEADER {
    USHORT e_magic;     // Сигнатура заголовка
    USHORT e_cblp;      // Кількість байт на одній сторінці файла
    USHORT e_cp;        // Кількість сторінок
    USHORT e_crlc;      // Релоки (Relocations)
    USHORT e_cparhdr;   // Розмір заголовка в параграфах
    USHORT e_minalloc;  // Мінімальні додаткові параграфи
    USHORT e_maxalloc;  // Максимальні додаткові параграфи
    USHORT e_ss;        // Початкове відносне значення регістра SS
    USHORT e_sp;        // Початкове значення регістра SP
    USHORT e_csum;      // Контрольна сума
    USHORT e_ip;        // Точка входу (Початкове значення регістра IP)
    USHORT e_cs;        // Початкове відносне значення регістра CS
    USHORT e_lfarlc;    // Адресса таблиці релокації
    USHORT e_ovno;      // Кількість оверлеїв
    USHORT e_res[4];    // Зарезервировано
    USHORT e_oemid;     // OEM ідентефікатор
    USHORT e_oeminfo;   // OEM інформація
    USHORT e_res2[10];  // Зарезервировано
    LONG   e_lfanew;    // Адресса PE загловка відносно початку файла
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

Оскільки ця частина залишена для мінімізації конфліктів з MS-DOS (FreeDos), розглядати її детально не має сенсу. В сучасних операційних системах використовується зазвичай тільки 2 поля цієї структури, а саме: e_magic та e_lfanew. Де e_magic – унікальна сигнатура файлу що займає 2 байта. В форматі PE вона завжди відповідає ASCII коду символів MZ; e_lfanew – являється зміщенням до сучасних PE заголовків відносно початку файлу.

Після структури IMAGE_DOS_HEADER зазвичай йде невелика кількість нулів, до початку Dos-заглушки (Dos-Stub). Це також частина файлу для зворотньої сумісності з операційними системами сімейства DOS. Це буквально частина коду під операційну систему DOS що виводить повідомлення про несумісність цього PE-файлу зі старою операційною системою.

Вже після цієї структури розпочинається «сучасна» частина формату.

PE-Заголовок (IMAGE_NT_HEADER [1]) – перша частина основного заголовку:

```
typedef struct _IMAGE_NT_HEADERS {
    DWORD           Signature;              // Сигнатура заголовка
    IMAGE_FILE_HEADER     FileHeader;
    IMAGE_OPTIONAL_HEADER OptionalHeader;
} IMAGE_NT_HEADERS, *PIMAGE_NT_HEADERS;
```

Сигнатура заголовка являється структурою з вкладеними підструктурами та сигнатурой. Сигнатура (Signature) займає 4 байти та має значення «PE\x00\x00», при записі в C-Style форматі. FileHeader та OptionalHeader являються вкладеними структурами, про них детальніше далі.

IMAGE_FILE_HEADER [1] – структура що описує базові характеристики файлу:

```
typedef struct _IMAGE_FILE_HEADER {
    WORD  Machine;  //  число що задає архітектуру процесора, має список можливих значень заданий в системі
    WORD  NumberOfSections; // кількість секцій у файлі, обмежено 96-секціями
    DWORD TimeDateStamp;  // дата і час створення файлу
    DWORD PointerToSymbolTable; // зміщення на таблицю символів, використовується для зберігання налагоджувальної інформації, в «релізах» зазвичай затирається нулями
    DWORD NumberOfSymbols;
    WORD  SizeOfOptionalHeader;  // розмір наступного заголовку
    WORD  Characteristics;  // характеристики файлу, як і Machine має діапазон можливих заданих в системі значень
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

IMAGE_OPTIONAL_HEADER [1] – не дивлячись на назву, заголовок обов’язковий та який містить в собі інформацію необхідну для завантаження файлу в пам’ять:

```
typedef struct _IMAGE_OPTIONAL_HEADER {
    WORD        Magic;
    BYTE        MajorLinkerVersion;
    BYTE        MinorLinkerVersion;
    DWORD       SizeOfCode;
    DWORD       SizeOfInitializedData;
    DWORD       SizeOfUninitializedData;
    DWORD       AddressOfEntryPoint;
```

```
    DWORD       BaseOfCode;
    DWORD       BaseOfData;
    DWORD       ImageBase;
    DWORD       SectionAlignment;
    DWORD       FileAlignment;
    WORD        MajorOperatingSystemVersion;
    WORD        MinorOperatingSystemVersion;
    WORD        MajorImageVersion;
    WORD        MinorImageVersion;
    WORD        MajorSubsystemVersion;
    WORD        MinorSubsystemVersion;
    DWORD       Win32VersionValue;
    DWORD       SizeOfImage;
    DWORD       SizeOfHeaders;
    DWORD       CheckSum;
    WORD        Subsystem;
    WORD        DllCharacteristics;
    DWORD       SizeOfStackReserve;
    DWORD       SizeOfStackCommit;
    DWORD       SizeOfHeapReserve;
    DWORD       SizeOfHeapCommit;
    DWORD       LoaderFlags;
    DWORD       NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY
            DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADE
R, *PIMAGE_OPTIONAL_HEADER;
```

AddressOfEntryPoint – точка входу в додаток; BaseOfCode і BaseOfData – початок коду та даних програми; ImageBase – базова адреса завантаження додатку, повинен бути кратним 64кб (зазвичай використовують значення 0x400000); SectionAligment і FileAligment – вирівнювання секцій у пам’яті та файлі; SizeOfHeaders – розмір заголовків в пам’яті після вирівнювання; NumberOfRvaAndSizes – кількість елементі в масиві «директорій», в актуальних ОС завжди рівний 16; DataDirectory – масив «директорій», містить посилання та розмір цих самих «директорій».

Далі в PE файлах йдуть таблиці секцій IMAGE_SECTION_HEADER [1]:

```
typedef struct _IMAGE_SECTION_HEADER {
    BYTE Name[IMAGE_SIZEOF_SHORT_NAME];
    union {
        DWORD PhysicalAddress;
        DWORD VirtualSize;
    } Misc;
    DWORD VirtualAddress;
    DWORD SizeOfRawData;
    DWORD PointerToRawData;
    DWORD PointerToRelocations;
    DWORD PointerToLinenumbers;
    WORD NumberOfRelocations;
    WORD NumberOfLinenumbers;
    DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

Де Name[IMAGE_SIZEOF_SHORT_NAME] – назва секції. Константа IMAGE_SIZEOF_SHORT_NAME має значення «8»; VirtualSize – розмір секції у

віртуальній пам’яті; SizeOfRawData – розмір секції у файлі; PointerToRawData – зміщення до початку секції; VirtualAddress – адреса секції у віртуальній пам’яті; Characteristics – права установлені на секцію, можливі значення наведені у таблиці 1.

Таблиця 1. Можливі значення поля Characteristics

IMAGE_SCN_CNT_CODE	0x00000020	Секція, що включає виконуваний код.
IMAGE_SCN_CNT_INITIALIZED_ DATA	0x00000040	Секція, що містить ініційовані данні.
IMAGE_SCN_CNT_UNINITIALIZED _DATA	0x00000080	Секція, що містить не ініційовані данні.
IMAGE_SCN_MEM_DISCARDABLE	0x02000000	Секція, що може бути видалена з пам’яті після створення процессу.
IMAGE_SCN_MEM_NOT_CACHED	0x04000000	Для секції повинні бути відключені механізми «кешування».
IMAGE_SCN_MEM_NOT_PAGED	0x08000000	Секція не може вигружатися у файл підкачки.
IMAGE_SCN_MEM_SHARED	0x10000000	Всі копії данного файлу(програми) можуть мати спільний екземпляр цієї секції, завичай використовується в DLL-бібліотеках.
IMAGE_SCN_MEM_EXECUTE	0x20000000	Секція, що має права на виконання.
IMAGE_SCN_MEM_READ	0x40000000	Секція, що має права на читання данних.
IMAGE_SCN_MEM_WRITE	0x80000000	Секція, що має права на запис

Варто зазначити, що секція з ресурсами повинна мати назву .rsrc, в іншому випадку данні з ресурсів не будуть завантажуватись стандартними методами.

Наступною структурою що йде у файлі є IMAGE_IMPORT_DESCRIPTOR[1].

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
        union {
                DWORD   Characteristics;
                        DWORD   OriginalFirstThunk;
                } DUMMYUNIONNAME;
                DWORD   TimeDateStamp;
                DWORD   ForwarderChain;
                DWORD   Name;
                DWORD   FirstThunk;
        }
IMAGE_IMPORT_DESCRIPTOR,*PIMAGE_IMPORT_DESCRIPTOR;
typedef struct _IMAGE_THUNK_DATA32 {
                union {
                DWORD ForwarderString;
                DWORD Function;
                DWORD Ordinal;
                DWORD AddressOfData;
              } u1;
             } IMAGE_THUNK_DATA32,*PIMAGE_THUNK_DATA32;
typedef struct _IMAGE_IMPORT_BY_NAME {
  WORD    Hint;
  BYTE    Name[1];
```

} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;

Детальний розбір цієї структури в процесі роботи – не потрібен. Оскільки буде генеруватись внутрішня IAT на основі даних завантажених в пам’ять. Подальший детальний розбір формату буде проводитись в процесі роботи.

1.2. Способи реалізації захисних систем

Розробка захистих систем [2] являється не тривіальною задачею, яка потребує значних ресурсів. При цьому дуже важливо правильно виставити пріорітети методів захисту та дотримутались загальних принципів розробки захисної системи:

-не один з механізмів захисту не повинен критично впливати на швидкість роботи ПЗ;

-механізм захисту повинен буди стійкий даже при повному розкритті алгоритмів його роботи;

-певна ізольованість захисних механізмів від оригінальної частини програмного забезпечення (але ні в якому разі не повна ізоляція оскільки це сильно снижує надійність механізмів захисту);

Способи вбудування захисних механізмів:

1) вставка механізму ліцензування на етапі розробки основного продукту;

2) перетворення основних виконуваних файлів (шифрування) до вигляду, при якому їх виконання неможливе, при цьому використовується «завантажувач» який відповідає за ліцензування та завантаження (після розшифровки) основних виконуваних файлів;

3) модифікація виконуваних файлів на низькому рівні, зашиття алгоритмів в ПЗ без модифікацій коду продукту;

4) комбінація методів.

Розглянемо особливості перших трьох методів.

Вставка механізму ліцензування на етапі розробки основного продукту – один із самих простих методів з погляду складності реалізації, при цьому потребує часу тих хто розробляє продукт, а відповідно впливає на кінцеву ціну продукту.

Перетворення основних виконуваних файлів – механізм який потребує додаткового додатка, а саме «завантажувача. Оптимальний варіант зі сторони затрат часу розробників, але надійність подібного захисту визиває сумніви, оскільки оригінальний образ майже завжди можна «дістати з пам’яті».

Третій метод являється оптимальним та використовується в більшості сучасних систем захисту, при цьому потребує мінімальних зусиль зі сторони розробників програмного забезпечення. Сам продукт що реалізує подібний метод складний в розробці і його складно підтримувати, потрібні спеціалісти високого рівня, але в сучасному світі має дуже мало життєздатних альтернатив.

Існуючі варіанти захисних систем

Більшість існуючих захисних[6] систем використовує метод модифікації виконуваних файлів. Їх можна поділити на два види:

- Захисні системи низького рівня захисту, без систем ліцензування, або з системами низького рівню захисту;
- Захисні системи високого рівня захисту, з допоміжними чи вбудованими системами ліцензування.

Захисні системи першого типу зазвичай мають основну характеристику – відсутність своєї системи віртуалізації коду, хоча є і виключення. До них можна віднести – PrivateExeProtector, Obsidian, ASProtect (не плутати з ASPack який являється просто пакером), tElock, SoftwareKey Protection PLUS. Кожен з цих програмних комплексів має або критичні недоліки в системі безпеки та більшість технік являються «відсталими», хоча Obsidian та SoftwareKey Protection PLUS мають систему віртуалізації – її корисність дуже сумнівна на фоні загальної слабкості захисту і критичних недоліків системи ліцензування.

На захисних системах другого типу варто зупинитись – детальніше. Прикладами таких систем являються - Enigma Protector, Oreans Themida/WinLicense, VMProtect. Програмне забезпечення захищене цими програмними комплексами практично не можливо зламати без наявності готової ліцензії, і навіть при наявності ліцензії це достатньо складно – зазвичай «злом» являється «стабілізацією» наявної ліцензії. Хоча і не варто виключати спеціалістів високого рівня що здібні відновити файл практично до оригінального стану. Головною складністю подобних систем являється підсистема віртуалізації коду[2] – це механізм що перетворює «нативний» код відповідної архітектури в «р-код» від віртуальний процесор згенерованої випадково архітектури. «Згенерована випадково» в даному випадку архітектура що залежить від конкретного продукту – деякі з продуктів можуть генерувати віртуальні машини що використовують декілька різних потоків інструкцій що переключаються по прихованим механізмам, до 128 регістрів, декілька різних блоків стеку, і все це змішане з великою кількістю криптографії та мутації коду. Але все ж у подібних систем є один великий недолік – вони критично вливають на продуктивність роботи програмного забезпечення. Віртуальна машина буквально – емулятор віртуального процесора що не використовує ніяких розширень команд (наприклад Intel VT-x/VT-d) і потребує дуже багато ресурсів.

При цих умовах віртуалізація один із самих надійних методів захисту. Розробка систем віртуалізації потребує значних ресурсів та висококласних спеціалістів. Приклад роботи віртуалізації коду (взята одна з перших версій VMProtect) на Рис. 2, Рис. 3 – оригінальний код.

Рис. 2. Приклад віртуалізованого коду

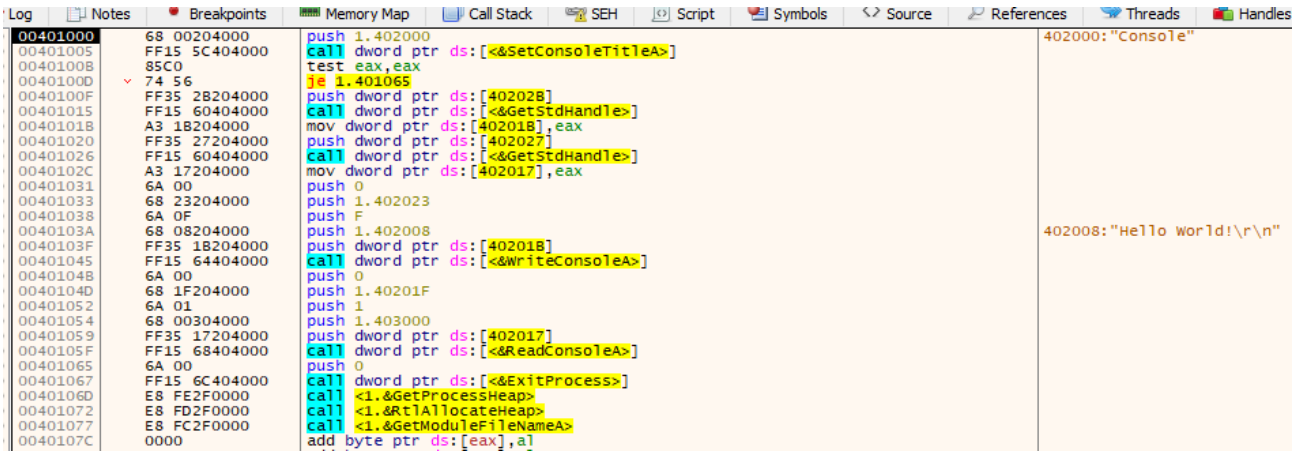

Рис. 3. Оригінальний код.

Як видно на прикладах код стає – абсолютно не читабельним, подальший аналіз не можливий без процесу «девіртуалізації» в замін на 40-кратне падіння швидкості роботи.

1.3. Постановка технічного завдання

На основі всього що було досліджено в процесі роботи складемо технічне завдання. Програмний комплекс повинен складатись з декількох частин:

- частина програми що буде виконуватись в середині ПЗ що ми захищаємо;
- програма що буде збирати першу частину і включати в оригінальне ПЗ, генерувати крипто-ключі;
- Web-сервер що буде відповідати за верифікації ліцензій.

Основна умова роботи ПЗ – програмне забезпечення не повинно бути зламано без наявної ліцензії, що являється оптимальним рішенням по співвідношенню затрачених ресурсів та надійності.

Розділ 2.
Реалізація технічного завдання

2.1. Клієнтська частина.

Клієнтська частина поділяється на 2 частини. Одна написана буде на мові програмування python і буде статична[8]. Включає в себе: шаблонізатор ассемблерного коду, генератор крипто-алгоритму, генератор ключів, відправку ключів на сервер. Друга частина буде динамічною і буде генеруватись за допомогою першої частини на основі шаблону. Розпочнемо саме з шаблону.
Він включає в себе внутрішню: IAT (Import Address Table)[6], підсистема прив'язки до «заліза», вікно авторизації, підсистема для зв'язку з веб-сервером, підсистема дешифровки коду, система верифікації коду. Ассемблерний код[7] збирається динамічно за допомогою інтерпретатора FASM.

Перш за все IAT:

```
mov eax, [fs:0x30]
mov eax, [eax+0Ch]
mov eax, [eax+14h]
mov eax, [eax]
mov eax, [eax]
mov ebx, [eax+10h]
mov eax, [ebx+3Ch]
mov eax, [eax+ebx+78h]
mov [exportDir+{{ CodeBase }}], eax
add eax, ebx
mov esi, [eax+20h]
mov [namePtr+{{ CodeBase }}], esi
```

```
mov edi, [eax+24h]
mov [ordPtr+{{ CodeBase }}], edi
mov eax, [esi+ebx]
add esi, ebx
add eax, ebx
push _GetProcAddress+{{ CodeBase }}
push eax
add edi, ebx
_check:
stdcall strcmpA
test eax,eax
jne _found
mov eax, [namePtr+{{ CodeBase }}]
add eax, 4
mov [namePtr+{{ CodeBase }}],eax
mov ecx,[ordPtr+{{ CodeBase }}]
add ecx,2
mov [ordPtr+{{ CodeBase }}], ecx
mov esi, [namePtr+{{ CodeBase }}]
add esi,ebx
add eax,ebx
mov eax,[eax]
add eax,ebx
push _GetProcAddress+{{ CodeBase }}
push eax
jmp _check
_found:
mov eax, [exportDir+{{ CodeBase }}]
```

add eax, ebx

mov ecx, ebx

add ecx, [eax+1Ch**]**

mov edx, [ordPtr+{{ CodeBase }}**]**

add edx, ebx

movzx eax, word ptr **edx**

mov ecx, [ecx+**eax***4**]**

add ecx, ebx

mov [@GetProcAddress+{{ CodeBase }}**], ecx**

mov [hKernel32+{{ CodeBase }}**], ebx**

stdcall [@GetProcAddress+{{ CodeBase }}**], [**hKernel32+{{ CodeBase }}**],** _LoadLibraryA+{{ CodeBase }}

mov [@LoadLibraryA+{{ CodeBase }}**], eax**

Зміщення для отримання структур директорій DIRECTORY, були вирахуванні на основі структури PEB, що завжди розміщена по зміщенню 0x30, відносно fs (fs:[0x30]). Далі була знаходится директорія DIRECTORY_ENTRY_EXPORT, після чого алгоритм отримує таблицю exportDir. I

Шаблон з циклічною генерацією:

{% for lib **in** IMPORT_LIB %}

stdcall [@LoadLibraryA+{{ CodeBase }}**],** {{ lib**[**"LibraryString"**]** }}+{{ CodeBase }}**, [**{{ lib**[**"hLibrary"**]** }}+{{ CodeBase }}**]**

mov [{{ lib**[**"hLibrary"**]** }}+{{ CodeBase }}**], eax**

{% end %}

{% for lib **in** IMPORT_LIB %}

{{ lib**[**"LibraryString"**]** }} db "{{ lib["LibraryName"] }}"**,** 0

{{ lib**[**"hLibrary"**]** }} dd **?,** 0

```
{% end %}
{% for function in IMPORT_FUNCTION %}
{{ function["NameString"] }} db "{{ function["SimbolicName"] }}", 0
@{{ function["SimbolicName"] }} dd ?
{% end %}
{% for function in IMPORT_FUNCTION[2:] %}
proc {{ function["SimbolicName"] }}
cmp [@{{ function["SimbolicName"] }}+{{ CodeBase }}],0
jnz flag_not_set{{ function["SimbolicName"] }}
stdcall [@GetProcAddress+{{ CodeBase }}], [{{ function["hLibrary"] }}+{{
CodeBase }}], {{ function["NameString"] }}+{{ CodeBase }}
mov [@{{ function["SimbolicName"] }}+{{ CodeBase }}], eax
flag_not_set{{ function["SimbolicName"] }}:
jmp [@{{ function["SimbolicName"] }}+{{ CodeBase }}]
endp
{% end %}
```

Відповідно цей блок генерує функції-перехідники та дані що використовуються під час пошуку необхідних функцій в таблиці експортів.

Данні шаблони генерують код такі блоки (частина для прикладу):

```
kernel32 db "kernel32.dll", 0
hKernel32 dd ?, 0
user32 db "user32.dll", 0
hUser32 dd ?, 0
_ExitProcess db "ExitProcess", 0
@ExitProcess dd ?
_CreateThread db "CreateThread", 0
```

@CreateThread dd **?**

proc ExitProcess

cmp [@ExitProcess+4227072**],**0

jnz flag_not_setExitProcess

stdcall [@GetProcAddress+4227072**], [**hKernel32+4227072**],** _ExitProcess+4227072

mov [@ExitProcess+4227072**], eax**

flag_not_setExitProcess**:**

jmp [@ExitProcess+4227072**]**

endp

proc CreateThread

cmp [@CreateThread+4227072**],**0

jnz flag_not_setCreateThread

stdcall [@GetProcAddress+4227072**], [**hKernel32+4227072**],**

_CreateThread+4227072

mov [@CreateThread+4227072**], eax**

flag_not_setCreateThread**:**

jmp [@CreateThread+4227072**]**

endp

Де 4227072 зміщення до вставленого сегменту.

Процедура отримання ключа для прив’язки до «заліза» - HWID:

proc GetSerialNumber

pusha

pushf

xor eax,eax

cpuid

mov dword[SourceKeyBuffer+{{ CodeBase }}**], ebx**

```
mov dword[SourceKeyBuffer+4+{{ CodeBase }}], edx
mov dword[SourceKeyBuffer+8+{{ CodeBase }}], ecx
xor eax,eax
inc eax
cpuid
mov dword[SourceKeyBuffer+12+{{ CodeBase }}], edx
mov dword[SourceKeyBuffer+16+{{ CodeBase }}], ecx
mov eax, 7
xor ecx, ecx
cpuid
mov dword[SourceKeyBuffer+20+{{ CodeBase }}], ecx
mov dword[SourceKeyBuffer+24+{{ CodeBase }}], edx
mov dword[SourceKeyBuffer+28+{{ CodeBase }}], ebx
stdcall SHA1, SourceKeyBuffer+{{ CodeBase }}, 100
stdcall  wsprintfA, RegistryKey+{{ CodeBase }}, sha_mask+{{ CodeBase }},
[SHA1_h0+{{ CodeBase }}], [SHA1_h1+{{ CodeBase }}], [SHA1_h2+{{ CodeBase
}}], [SHA1_h3+{{ CodeBase }}], [SHA1_h4+{{ CodeBase }}]
add     esp,7*4
popf
popa
ret
endp
```

Вивід вікна авторизації для вводу даних за допомогою CreateWindowEx:

```
stdcall GetModuleHandleA,0
mov [wc.hInstance+{{ CodeBase }}],eax
stdcall LoadIconA,0,IDI_APPLICATION
mov [wc.hIcon+{{ CodeBase }}],eax
```

```
stdcall LoadCursorA,0,IDC_ARROW
mov [wc.hCursor+{{ CodeBase }}],eax
stdcall RegisterClassA,wc+{{ CodeBase }}
cmp eax,0
je error
mov [wc.lpfnWndProc+{{ CodeBase }}], WindowProc+{{ CodeBase }}
stdcall CreateWindowExA,0,class+{{ CodeBase }},title+{{ CodeBase
}},WS_VISIBLE,128,128,256,196,0,0,[wc.hInstance+{{ CodeBase }}],0
cmp eax,0
je error
mov [hwnd+{{ CodeBase }}],eax
msg_loop:
stdcall GetMessageA,msg+{{ CodeBase }},0,0,0
cmp eax,0
je end_loop
stdcall IsDialogMessageA,[hwnd+{{ CodeBase }}],msg+{{ CodeBase }}
cmp eax,0
jne msg_loop
stdcall TranslateMessage,msg+{{ CodeBase }}
stdcall DispatchMessageA,msg+{{ CodeBase }}
jmp msg_loop
```

Віконна процедура:

```
proc WindowProc hwnd,wmsg,wparam,lparam
  push ebx esi edi
  cmp [wmsg],WM_CREATE
  je .wmcreate
  cmp [wmsg],WM_COMMAND
```

```
  je .wmcommand
  cmp [wmsg],WM_DESTROY
  je .wmdestroy
.defwndproc:
  stdcall DefWindowProcA,[hwnd],[wmsg],[wparam],[lparam]
  jmp .finish
.wmcreate:
  stdcall CreateWindowExA,0,ClassButton+{{ CodeBase }},LoginButtonText+{{
CodeBase }},WS_VISIBLE+WS_CHILD+
BS_PUSHBUTTON+WS_TABSTOP,20,100,100,40,[hwnd],1000,[wc.hInstance+{{
CodeBase }}],0
  stdcall CreateWindowExA,0,ClassButton+{{ CodeBase }},ExitButtonText+{{
CodeBase }},WS_VISIBLE+WS_CHILD+
BS_PUSHBUTTON+WS_TABSTOP,130,100,100,40,[hwnd],1001,[wc.hInstance+{{
CodeBase }}],0
  stdcall CreateWindowExA,0,classe+{{ CodeBase
}},0,WS_VISIBLE+WS_CHILD+WS_BORDER+WS_TABSTOP+ES_AUTOHSC
ROLL,10,50,230,20,[hwnd],1002,[wc.hInstance+{{ CodeBase }}],0
  mov [hwndKeyEdit+{{ CodeBase }}],eax
  stdcall CreateWindowExA,0,classe+{{ CodeBase }},RegistryKey+{{ CodeBase
}},WS_VISIBLE+WS_CHILD+WS_BORDER+WS_TABSTOP+ES_AUTOHSCR
OLL+ES_READONLY,10,25,230,20,[hwnd],1004,[wc.hInstance+{{ CodeBase }}],
0
  mov [hwndHWIDEdit+{{ CodeBase }}],eax
  stdcall SetFocus,[hwndKeyEdit+{{ CodeBase }}]
  jmp .finish
.wmcommand:
  cmp [wparam],1000
```

```
  je .LoginButton
  cmp [wparam],1001
  je .ExitButton
  jmp .finish
.LoginButton:
  mov word[LoginText-2+{{ CodeBase }}], 0x0A0D
  stdcall SendMessageA,[hwndKeyEdit+{{ CodeBase
}}],WM_GETTEXT,100,LoginText+{{ CodeBase }}
  stdcall WebRequest, hostname+{{ CodeBase }}, WebResultBuffer+{{ CodeBase
}}
  stdcall DestroyWindow, [hwnd]
  stdcall DecodeSelectionAndJMP, WebResultBuffer+{{ CodeBase }}
  jmp .finish
.ExitButton:
  stdcall ExitProcess, 0
  jmp .finish
.wmdestroy:
  stdcall PostQuitMessage,0
  mov eax,0
.finish:
  pop edi esi ebx
  ret
endp
```

Віконна процедура – це обробних всіх поїй створеного вікна.

Вигляд вікна авторизації з ключем прив’язки на Рис. 4.

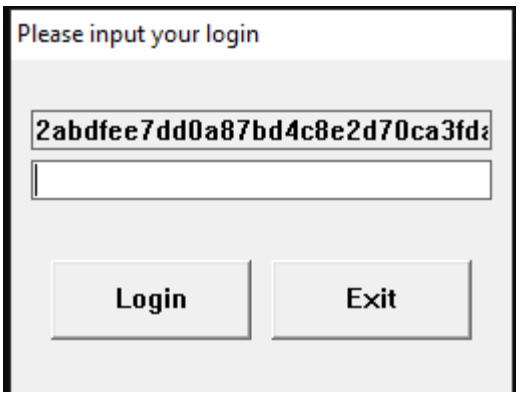

Рис. 4. Вікно авторизації

Підсистема зв’язку з Web-сервером[12]:

```
proc WebRequest link, WebResultBuffer
        pusha
        pushf
        stdcall WSAStartup, 0101h, wsadata+{{ CodeBase }}
        stdcall  gethostbyname, [link]
        or     eax, eax
        jz     bad_hostname
              virtual at eax
               .host hostent
              end virtual
        mov    eax, [.host.h_addr_list]
        mov    eax, [eax]
        mov    eax, [eax]
        mov     [saddr.sin_addr+{{ CodeBase }}], eax
        mov    al,00
        mov    ah,80
        mov     [saddr.sin_port+{{ CodeBase }}], ax
        mov     [saddr.sin_family+{{ CodeBase }}], AF_INET
        stdcall  socket, AF_INET, SOCK_STREAM, IPPROTO_TCP
```

```
        mov     [hSock+{{ CodeBase }}], eax
        xchg    eax, esi
        stdcall  connect, esi, saddr+{{ CodeBase }}, sizesaddr+{{ CodeBase }}
        stdcall strlenA, sender+{{ CodeBase }}
        stdcall  send, esi, sender+{{ CodeBase }},eax,0
        stdcall  recv, esi, [WebResultBuffer], 1024, 0
        .connectSucceeded:
        stdcall closesocket,esi
        stdcall WSACleanup
        popf
        popa
        ret
        bad_hostname:
        stdcall ExitProcess,0
        ret
endp
```

Підсистема дешифровки коду[9, 10], також використовує механізм шаблонізації, оскільки кожна секція файлу шифрується окремо, в якості перевірки цілісності дешифрованої секції використовується перевірка контрольої суми CRC32:

```
proc DecodeSelectionAndJMP DecoderKey
{% for section in EncodeSelecion %}
mov ebx, {{ section["DecodeAddr"]-1 }}
mov ecx, {{ section["DecodeSize"] }}
mov edx, [DecoderKey]
EncoderLoop{{section["loop_iteration"]}}:
```

```
xor eax,eax
mov al, byte [ebx+ecx]
{{ Encoder }}
mov byte [ebx+ecx], al
loop EncoderLoop{{section["loop_iteration"]}}
stdcall calc_CRC32, {{ section["DecodeAddr"] }}, {{ section["DecodeSize"] }}, -1,
1
cmp eax, {{ section["crc32"] }}
jne @Decoder_Exit
{% end %}
jmp {{ OriginalEP-CodeBase }}
@Decoder_Exit:
stdcall ExitProcess, 0
endp
```

Допоміжні дані(в основном структури та текстові рядки для роботи віконних процедур та частини що займається відправкою запитів на сервер):

```
wc WNDCLASS 0,WindowProc+{{ CodeBase }}
,0,0,0,0,0,COLOR_BTNFACE+1,0,class+{{ CodeBase }}
class db 'LicSystem',0
title db 'Please input your login',0
ClassButton db 'BUTTON',0
classe db 'EDIT',0
classs db 'STATIC',0
LoginButtonText db 'Login',0
ExitButtonText db 'Exit',0
hwnd dd ?
hwndKeyEdit dd ?
```

```
hwndHWIDEdit dd ?
hwndc dd ?
sha_mask db '%.8x%.8x%.8x%.8x%.8x',0
SourceKeyBuffer rb 100
IPPROTO_TCP  = 6
wsadata WSADATA
```

Допоміжні функції – упустимо, вони являються PIC реалізацією стандартних алгоритмів (Додаток Б).

Відключення алгоритму ASLR[11]:

```
def Disable_Aslr(pe):
   print("[*] Check ASLR")
   IMAGE_DLL_CHARACTERISTICS_DYNAMIC_BASE  = 0x40
   if (pe.OPTIONAL_HEADER.DllCharacteristics &
IMAGE_DLL_CHARACTERISTICS_DYNAMIC_BASE):
      pe.OPTIONAL_HEADER.DllCharacteristics &=
~IMAGE_DLL_CHARACTERISTICS_DYNAMIC_BASE
      print("ASLR disabled")
   else:
      print("ASLR not enabled")
```

Шаблонізатор же працює на основі публічно доступних на модифікованих бібліотек tornado.template та pefile(модифікації в додатку В) за авторством «Ero Carrera», а також розроблена бібліотека register_operation для реалізації базових бінарних операцій над потоками даних. Всі бібліотеки розповсюджуються під ліцензією GNU GPL. Однією із самих важливих частин являється генератор алгоритму шифрування, що генерується кожен раз унікальний. Про стійкість цього алгоритму – далі.

Генератор:

```
def GenerateDecoder(Key):
    encode_instructions = ["add", "sub"]
    decoder = ""
    next_xor = True
    for i in range(len(Key)):
        if next_xor:
            instruction = "xor"
            next_xor = False
        else:
            instruction = random.choice(encode_instructions)
            next_xor = True
        decoder += instruction + " al, byte[edx+"+str(i)+"]\n"
    decoder = decoder.split("\n")
    decoder = filter(None, decoder)
    return decoder
```

Результат роботи блоку:

```
mov al, byte [ebx+ecx]
xor al, byte[edx+0]
sub al, byte[edx+1]
…
add al, byte[edx+N]
mov byte [ebx+ecx], al
```

Далі алгоритм достатньо простий, до виконуваного файлу дописується секція, а секція з оригінальним кодом шифрується. Точка входу міняється на початок нової секції, куди записується сгенерований шаблонізатором та

інтерпретований код. Разом з цим на сервер дешифровки передаються ключі що необхідні для дешифровки оригінального коду. Блок коду що відповідає за роботу шаблонізатора:

```
Template(open("ProtectorSelection.tpl.asm", "r").read()).generate(
    xor_len=pe.sections[0].Misc_VirtualSize,
    Encoder='\n'.join(map(str, decoder)),
    OriginalEP = pe.OPTIONAL_HEADER.ImageBase +
pe.OPTIONAL_HEADER.AddressOfEntryPoint,
    CodeBase = pe.OPTIONAL_HEADER.ImageBase +
pe.sections[SectionMap[".xcode"]].VirtualAddress,
    EncodeSelecion = EncodeSelecion,
    DecodeKey=key[::-1],
    IMPORT_LIB = IMPORT_LIB,
    IMPORT_FUNCTION = IMPORT_FUNCTION,
    build_id = build_id
```

Вставка інтерпретованого коду в секцію створену нашим продуктом та зміна точки входу:

```
pe.data_replace(offset=pe.sections[SectionMap[".xcode"]].PointerToRawData,
new_data=ProtectorSelection)
  pe.OPTIONAL_HEADER.AddressOfEntryPoint =
pe.sections[SectionMap[".xcode"]].VirtualAddress
```

2.2. Серверна частина

Реалізація серверної частини базується на основі бібліотек socket [8, 12] та sqlite3. Sqlite – це драйвер для роботи з базами даних одно-іменного формату. Socket – реалізація «сирих» синхронних сокетів для мультиплатформи:

```
import socket
import sqlite3
def send_answer(conn, data=""):
    data = data.encode("utf-8")
    conn.send(data)
def parse(conn, addr):
    data = b""
    while not b"\r\n" in data:
        tmp = conn.recv(1024)
        if not tmp:
            break
        else:
            data += tmp
    if not data:
        return
    request_data = str(data).splitlines()
    biuld_id = request_data[0]
    key = request_data[1]
    login = request_data[2]
    sql_connect = sqlite3.connect(r'./Decoders.db')
    c = sql_connect.cursor()
    Licence = False
```

```
    for row in c.execute("SELECT * FROM `software_users` WHERE `software_users`.`login` LIKE '"+login+"' AND `software_users`.`hwid` LIKE '"+key+"'"):
        Licence = True
    if Licence:
        for row in c.execute("SELECT * FROM `builds` WHERE `builds`.`build_id` LIKE '"+biuld_id+"'"):
            send_answer(conn, data=row[2][::-1]) #Функція видачі ключа, при успішній перевірці ліцензії
    else:
        send_answer(conn, data="Error")
    sql_connect.commit()
    sql_connect.close()

sock = socket.socket()
sock.bind(("", 80))
sock.listen(5)
try:
    while 1:
        conn, addr = sock.accept()
        print("New connection from " + addr[0])
        parse(conn, addr)
        conn.close()
finally:
    sock.close()
```

Структура бази даних sqlite наведена у Таблиці 2.

Таблиця 2. Структура бази даних.

Ім’я таблиці/поля	Тип поля	Схема
builds		CREATE TABLE `builds` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `build_id` TEXT, `decode_key` TEXT)
id	INTEGER	"id" INTEGER PRIMARY KEY AUTOINCREMENT
build_id	TEXT	"build_id" TEXT
decode_key	TEXT	"decode_key" TEXT
software		"id" INTEGER PRIMARY KEY AUTOINCREMENT
id	INTEGER	"id" INTEGER PRIMARY KEY AUTOINCREMENT
name	TEXT	"name" TEXT
autor_name	TEXT	"autor_name" TEXT
software_users		CREATE TABLE "software_users" (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `login` TEXT, `hwid` TEXT, `software` INTEGER)
id	INTEGER	"id" INTEGER PRIMARY KEY AUTOINCREMENT
login	TEXT	"login" TEXT
hwid	TEXT	"hwid" TEXT
software	INTEGER	"software" INTEGER

Розділ 3.
Тестування на стійкість

3.1. Тестування програмного комплексу на стійкість

Перш за все варто поговорити про довжину ключа, яку можна змінювати. За основу візьмемо стандартну довжину ключа в 50 символів, кожен із символів це байт, тобто він може прийняти одне з 255 значень. Правило розміщення говорить що 225^50 і буде максимальною можливою кількістю паролів при цій довжині паролю. При цьому одна ітерація обчислення займає приблизно 0.004 секунди на CPU з використанням одного швидкого ядра (за умови що кількість шифрованих даних 4 кб, залежність лінійна). Для обчислення часу була написана маленька програма(python):

```
iter_count = (255**50)
second = iter_count*0.004
hours = second/3600
days = hours/24
years = days/365
print 'iter_count = {:.0f}'.format(iter_count)
print 'second = {:.0f}'.format(second)
print 'hours = {:.0f}'.format(hours)
print 'days = {:.0f}'.format(days)
print 'years = {:.0f}'.format(years)
```

Її результат при даних вхідних параметрах(частина результатів – упущена):

iter_count=212328856909704353464661758328047860872749292494068666257620600559023069932316137050752721197676947016471117980675709337626
years=269316155390289653929545575876070196951124931462954098204803552882132624021730012248682773569149061173891563520

Відповідно криптографію можна назвати стійкою, за умови що стійкість визначається неможливістю провести атаку підбору ключа за лінійний проміжок часу.

Іншою можливістю атаки може бути реалізація підбору даних ініціалізації генератора псевдовипадкових чисел. При цьому не можливо підрахувати чи буде достатньо 75 відомих значень (при 50 значеннях ключа та 25 випадкових елементах алгоритму) для проведення атаки, оскільки подібні атаки зазвичай потребують великі масиви генерованих даних. Ця думка була підкріплена результатами тестування на стійкість в реальних умовах з реальними спеціалістами з реверс-інжинірінгу які бажали залишитись анонімними. При цьому тестування приводилось при вхідних даних с 20 символьним ключем що використовував набір символів ASCII англійського алфавіту, результат:

iter_count=133674945388437340315549622599682
second=53469978155374941544613150720
hours=1485277170982637226950656
days=618865487909432133156864
years=1695521884683375738880

Особлива подяка «OKOB», члену спільноти «R0 Crew» з ресурсу R0 CREW Forum за вклад в «бойове тестування» програмного комплексу[4, 5].

Висновки

В процесі роботи було досягнуто основної цілі – розробка продукту що при мінімальній взаємодії зі сторони розробника оригінального ПЗ здібний забезпечити надійний захист. Під час реалізації технічного завдання було розроблено декілька цікавих алгоритмів, в особливості алгоритм генерації простої криптографії на основі базових операцій. Також достатньо цікавою задачею була розробка коду що буде виконуватись в контексті скомпільованого виконуваного файлу. Був розроблений механізм для включення виконуваного коду в готові виконувані файли на основі модифікованої бібліотеки та шаблонізатора, що зазвичай використовується для веб-застосунків.

Було проведено тестування на стійкість в ході якого була доведена практична неефективність атаки на додатки захищені системою ліцензування. Варто додати що навіть при використанні самих сучасних систем перебору паролів перебір буте тривати десятки тисяч років.

На основі вище сказаного можна сказати що технічне завдання виконано повністю.

Список використаних джерел

1. MSDN Windows API index [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/

2. Каплун В.А, Дмитришин О. В., Баришев Ю. В. «Захист програмного забезпечення», Вінниця ВНТУ 2014

3. Packers & Protectors [Електронний ресурс] – Режим доступу: https://forum.tuts4you.com/files/category/52-packers-protectors/

4. SOLS - Simle open license system [Електронний ресурс] – Режим доступу: https://forum.reverse4you.org/t/sols-simle-open-license-system-crackme/1161

5. Работа с PEB и TEB [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/187226/

6. PE (Portable Executable): На странных берегах [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/266831/

7. «Assembler для DOS, Windows и UNIX» / Зубков Сергей Владимирович. - 3-е изд., стер. - М. : ДМК Пресс ; СПб. : Питер, 2004. - 608 с.

8. «Byte of python» [Електронний посібник] – Режим доступу: https://github.com/swaroopch/byte-of-python

9. Брюс Шнайер. «Прикладная криптография». 2016.

10. В. В. Ященко. «Введение в криптографию». Москва, МЦНМО 2012.

11. Mark E. Russinovich David A. Solomon. Windows® Internals Fourth Edition.

12. В. Олифер, Н. Олифер «Компьютерные сети. Принципы, технологии, протоколы»

13. Cord cutters [Електронний ресурс] – Режим доступу: https://www.cordcuttersnews.com/piracy-is-expected-to-cause-12-5-billion-in-losses-by-2024-this-tech-company-is-addressing-it/

ДОДАТОК А

Тези доповіді на IV Всеукраїнській науково-практичній конференції здобувачів вищої освіти та молодих науковців «Прикладні аспекти інформаційного забезпечення та обґрунтування технічних і управлінських рішень»

СУЧАСНІ ПІДХОДИ ДОЗАХИСТУ ПРОГРАМНОГОЗАБЕЗПЕЧЕННЯ

Бєліч А. О.,студент

Кот В. В., ктн., доц. кафедри інформатики та прикладної математики

Рівненський державний гуманітарний університет

Досвід останніх років показав, що надійність і функціональність комп'ютерних систем залежить від якості програмного забезпечення (ПЗ), що входить в їх склад. Однак розробка якісного ПЗ потребує серйозних капіталовкладень і відповідно пов'язана з ризиками втрати капіталу. Для зменшення ризиків необхідно мінімізувати «піратське» використання ПЗ. Розробка захисних систем являється не тривіальною задачею, яка потребує значних ресурсів. При цьому дуже важливо правильно виставити пріоритети методів захисту та дотримуватись загальних принципів розробки захисної системи:

- не один з механізмів захисту не повинен критично впливати на швидкість роботи ПЗ;
- механізм захисту повинен буди стійкий навітьпри повному розкритті алгоритмів його роботи;
- певна ізольованість захисних механізмів від основної частини ПЗ (але ні в якому разі не повна ізоляція оскільки це сильно знижує надійність механізмів захисту);

•контроль наявності ліцензії повинен відбуватись протягом всіхетапів роботи основної програми. Відповідно пріоритетів визначають наступні способи вбудування захисних механізміву ПЗ:

•вставка механізму ліцензування в на етапі розробки основного продукту;

•перетворення основних виконуваних файлів (шифрування) до вигляду при якому їх виконання неможливе, при цьому використовується завантажувач який відповідає за ліцензування та завантаження (після розшифровки) основних виконуваних файлів;

•модифікація виконуваних файлів на низькому рівні, зашиття алгоритмів в ПЗ без модифікацій коду продукту;

•комбінація методів;Для операційних систем сімейства Windowsпрацюючих на архітектурі x86(x86_64)дані способи розширюютьсяна такі види:

•захист від статичного дізасемблювання: обфускація, маніпуляція PEзаголовками, криптографічні методи захисту коду;

•захист від динамічного аналізу: хукі, методи операційної системи, пошук відлагоджувачів різними методами, антивідлагоджуючі механізми;

•захист від технологій заснованих на дампі в процесі виконання: обмеження доступу до сторінок пам’яті, модифікація механізму релокації, використання коду що може самомодифікуватись;

•загальні методи які вносять проблеми при всіх видах аналізу: віртуалізація, емуляція складних процесів багатозадачності.

Найбільш ефективним методом являється комбінація всіх механізмів захисту, але потрібно встановити рамки: вносити критичні модифікації в код ПЗ на етапірозробки недопустимо, оскільки це сильно ускладнює розробку. Відповідно методи захисту повинні буди відносно зовнішніми тав цьому

випадку захисний механізм виглядає так: зовнішня захисна система із мінімальною підтримкою зі сторони розробника основного продуктуяка влаштовується у виконувані файли основного продуктумодифікуючи код для неможливості його виконання без захисного механізму.

В ході виконання магістерської роботи запропоновано розробити програмнийкомплексна основі вище названих критеріїв. Весь комплекс ділиться на 3 частини: веб-сервер, протектор та механізм(код) що буде влаштовуватись у виконуваний файл базового ПЗ.

Веб-сервер –написаний на мові python, використовується свій протокол зв'язку поверх базового інтерфейсу «Socket»та СУБД sqlite.

Протектор –програма що займається процесом інтеграції коду у виконуваний файл. Для асемблювання використовується FASM що розповсюджується під ліцензією GNUGPL.В якості допоміжного елементу для шаблонізації коду використано частину фреймворку tornado(якщо точніше то модуль Template). Також на цьому етапі формується майбутній базовий криптоалгоритм на основі великої кількості зв'язаних базових операцій xor, add, sub. Код оригінального виконуваного файлу шифрується за допомогою цього алгоритму і буде розшифрований тільки після верифікації ліцензії та отримання ключа з веб-сервера.

Інтегрований код –код написаний на асемблері розрахований на влаштування у виконувані файли, містить механізми незалежної IAT, механізми зв'язку з мережею та механізми верифікації ліцензії для прив'язки до конкретної комп'ютерної системи. Даний код не впливає на виконання основної частини програми за виключенням спеціально промаркерованних розробниками моментів та регулярної верифікації ліцензії в процесі роботи.

Список використаних джерел

1.Microsoft Developer Network[Електронний ресурс] - https://docs.microsoft.com/en-us/

2.Каплун В.А, Дмитришин О. В., БаришевЮ. В. Захист програмного забезпечення, Вінниця ВНТУ 2014

3.Tuts 4 You[Електронний ресурс]- https://tuts4you.com/

4.ExeLab[Електронний ресурс] - https://exelab.ru/

5.R0 CREW Forum [Електронний ресурс] - https://forum.reverse4you.org/

ДОДАТОК Б

Допоміжні функції

Допоміжні функції що не були включені в основну роботу через об’єм і типовість, але були модифіковані для базо-незалежного використання.

Функція розрахунку контрольної суми CRC32

```
proc calc_CRC32 lData:dword, dLen:dword, dCRC32:dword, dFlag:dword
        push    ebx ecx edx esi

        mov     edx,[dCRC32]
        mov     esi,[lData]
        mov     ecx,[dLen]
        xor     eax,eax
calc_crc32:
        lodsb
        mov     ebx,edx
        and     ebx,0FFh
        xor     bl,al
        shr     edx,8
        and     edx,0FFFFFFh
        push    ecx
        mov     ecx,8
do_polynom:
        shr     ebx,1
        jnc     @f
        xor     ebx,0EDB88320h
@@:
        loop    do_polynom
        pop     ecx
        xor     edx,ebx
        loop    calc_crc32
        mov     eax,edx
        cmp     [dFlag],1
        jne     @f
        xor     eax,-1
@@:
        pop     esi edx ecx ebx
        ret
```

endp

Функція розрахунку контрольної суми SHA-1, а також допоміжні глобальні змінні:

```
proc    SHA1 lpData:DWORD, dSize:DWORD
        locals
                len     dd ?
                padding dd ?
                index   dd ?
        endl
        pusha
        mov     [SHA1_h0+{{ CodeBase }}],0x67452301
        mov     [SHA1_h1+{{ CodeBase }}],0xEFCDAB89
        mov     [SHA1_h2+{{ CodeBase }}],0x98BADCFE
        mov     [SHA1_h3+{{ CodeBase }}],0x10325476
        mov     [SHA1_h4+{{ CodeBase }}],0xC3D2E1F0

        mov     eax,[dSize]
        push    eax
        shr     eax,6
        mov     [len],eax
        pop     eax

        and     eax,3Fh
        mov     [padding],eax
        mov     [index],0
.main_loop:
        mov     eax,[index]
        cmp     eax,[len]
        je      .main_done

        shl     eax,6
        mov     esi,[lpData]
        add     esi,eax
        mov     edi,SHA1_Buff+{{ CodeBase }}
        mov     ecx,16
        rep     movsd
```

```
        stdcall SHA1_BE
        stdcall SHA1_Calc

        inc     [index]
        jmp     .main_loop
.main_done:
        xor     eax,eax
        mov     edi,SHA1_Buff+{{ CodeBase }}
        mov     ecx,16
        rep     stosd

        mov     eax,[index]
        shl     eax,6
        mov     esi,[lpData]
        add     esi,eax
        mov     edi,SHA1_Buff+{{ CodeBase }}
        mov     ecx,[padding]
        rep     movsb
        mov     al,80h
        stosb

        stdcall SHA1_BE
        cmp     [padding],56
        jae     @f

        mov     eax,[dSize]
        shl     eax,3
        mov     dword [SHA1_Buff+60+{{ CodeBase }}],eax

        stdcall SHA1_Calc
        jmp     .sha1_done
@@:
        stdcall SHA1_Calc

        mov     edi,SHA1_Buff+{{ CodeBase }}
```

```
        xor     eax,eax
        mov     ecx,15
        rep     stosd

        mov     eax,[dSize]
        shl     eax,3
        stosd

        stdcall SHA1_Calc

.sha1_done:
        popa
        ret
endp

proc    SHA1_BE
        pusha

        mov     ecx,16
        mov     esi,SHA1_Buff+{{ CodeBase }}
        mov     edi,esi
@@:
        lodsd
        bswap   eax
        stosd
        loop    @b

        popa
        ret
endp

proc    SHA1_Calc
        pusha

        mov     eax,[SHA1_h0+{{ CodeBase }}]
        mov     [SHA1_a+{{ CodeBase }}],eax
```

```
        mov     eax,[SHA1_h1+{{ CodeBase }}]
        mov     [SHA1_b+{{ CodeBase }}],eax
        mov     eax,[SHA1_h2+{{ CodeBase }}]
        mov     [SHA1_c+{{ CodeBase }}],eax
        mov     eax,[SHA1_h3+{{ CodeBase }}]
        mov     [SHA1_d+{{ CodeBase }}],eax
        mov     eax,[SHA1_h4+{{ CodeBase }}]
        mov     [SHA1_e+{{ CodeBase }}],eax

        xor     ecx,ecx ; i
.cycle_loop:
        mov     eax,ecx
        shl     eax,2

        cmp     ecx,16
        jae     @f

        mov     ebx,dword [SHA1_Buff+eax+{{ CodeBase }}]
        mov     dword [SHA1_W+eax+{{ CodeBase }}],ebx
        jmp     .@1
@@:
        mov     eax,ecx
        sub     eax,3
        shl     eax,2
        mov     ebx,dword [SHA1_W+eax+{{ CodeBase }}]

        mov     eax,ecx
        sub     eax,8
        shl     eax,2
        xor     ebx,dword [SHA1_W+eax+{{ CodeBase }}]

        mov     eax,ecx
        sub     eax,14
        shl     eax,2
        xor     ebx,dword [SHA1_W+eax+{{ CodeBase }}]
```

```
        mov     eax,ecx
        sub     eax,16
        shl     eax,2
        xor     ebx,dword [SHA1_W+eax+{{ CodeBase }}]

        rol     ebx,1

        mov     eax,ecx
        shl     eax,2
        mov     dword [SHA1_W+eax+{{ CodeBase }}],ebx
.@1:
        mov     edx,[SHA1_a+{{ CodeBase }}]
        rol     edx,5

        cmp     ecx,20
        jae     @f
        mov     eax,[SHA1_b+{{ CodeBase }}]
        and     eax,[SHA1_c+{{ CodeBase }}]
        mov     ebx,[SHA1_b+{{ CodeBase }}]
        not     ebx
        and     ebx,[SHA1_d+{{ CodeBase }}]
        or      eax,ebx
        add     edx,0x5A827999
        jmp     .@2
@@:
        cmp     ecx,40
        jae     @f
        mov     eax,[SHA1_b+{{ CodeBase }}]
        xor     eax,[SHA1_c+{{ CodeBase }}]
        xor     eax,[SHA1_d+{{ CodeBase }}]
        add     edx,0x6ED9EBA1
        jmp     .@2
@@:
        cmp     ecx,60
        jae     @f
        mov     eax,[SHA1_b+{{ CodeBase }}]
```

```
        and     eax,[SHA1_c+{{ CodeBase }}]
        mov     ebx,[SHA1_b+{{ CodeBase }}]
        and     ebx,[SHA1_d+{{ CodeBase }}]
        or      eax,ebx
        mov     ebx,[SHA1_c+{{ CodeBase }}]
        and     ebx,[SHA1_d+{{ CodeBase }}]
        or      eax,ebx
        add     edx,0x8F1BBCDC
        jmp     .@2
@@:
        mov     eax,[SHA1_b+{{ CodeBase }}]
        xor     eax,[SHA1_c+{{ CodeBase }}]
        xor     eax,[SHA1_d+{{ CodeBase }}]
        add     edx,0xCA62C1D6
.@2:
        add     edx,eax
        add     edx,[SHA1_e+{{ CodeBase }}]

        mov     eax,ecx
        shl     eax,2
        add     edx,[SHA1_W+eax+{{ CodeBase }}]

        mov     eax,[SHA1_d+{{ CodeBase }}]
        mov     [SHA1_e+{{ CodeBase }}],eax

        mov     eax,[SHA1_c+{{ CodeBase }}]
        mov     [SHA1_d+{{ CodeBase }}],eax

        mov     eax,[SHA1_b+{{ CodeBase }}]
        rol     eax,30
        mov     [SHA1_c+{{ CodeBase }}],eax

        mov     eax,[SHA1_a+{{ CodeBase }}]
        mov     [SHA1_b+{{ CodeBase }}],eax

        mov     [SHA1_a+{{ CodeBase }}],edx
```

```
        inc     ecx
        cmp     ecx,80
        jne     .cycle_loop

        mov     eax,[SHA1_a+{{ CodeBase }}]
        add     [SHA1_h0+{{ CodeBase }}],eax
        mov     eax,[SHA1_b+{{ CodeBase }}]
        add     [SHA1_h1+{{ CodeBase }}],eax
        mov     eax,[SHA1_c+{{ CodeBase }}]
        add     [SHA1_h2+{{ CodeBase }}],eax
        mov     eax,[SHA1_d+{{ CodeBase }}]
        add     [SHA1_h3+{{ CodeBase }}],eax
        mov     eax,[SHA1_e+{{ CodeBase }}]
        add     [SHA1_h4+{{ CodeBase }}],eax

        popa
        ret
endp
fill db 2 dup(0)
SHA1_h0         dd ?
SHA1_h1         dd ?
SHA1_h2         dd ?
SHA1_h3         dd ?
SHA1_h4         dd ?

SHA1_a          dd ?
SHA1_b          dd ?
SHA1_c          dd ?
SHA1_d          dd ?
SHA1_e          dd ?
SHA1_W          rd 80
SHA1_Buff       rb 64
```

Функції для отримання довжини рядків та функція для порівняння рядків кодування ASCII:

```
proc     strcmpA,szOne,szTwo
   pusha
   pushf
   mov esi, [szOne]
   mov edi, [szTwo]
   stdcall strlenA, edi
   inc eax
   mov ecx, eax
   repe cmpsb
   je @cmpValid
   popf
   popa
   mov eax, 0
   ret
   @cmpValid:
   popf
   popa
   mov eax, 1
   ret
endp

proc strlenA lData:dword
   push edi
   mov edi,[lData]
   xor eax,eax
   @1:
   inc eax
   cmp byte[eax+edi],0
   jnz @1
   pop edi
   ret
endp
```

ДОДАТОК В

Модифікації для бібліотеки pefile

```
def do_xor(value_in, xor_val):
    xor = value_in ^ xor_val
    if (xor >= 256) or (xor < 0):
        xor = "{:02x}".format(xor & 0xffffffff)[-2:]
        xor = int(xor,16)
    return xor

def do_add(value_in, add_val):
    add = value_in + add_val
    if (add >= 256) or (add < 0):
        add = "{:02x}".format(add & 0xffffffff)[-2:]
        add = int(add, 16)
    return add

def do_sub(value_in, sub_val):
    sub = value_in - sub_val
    if (sub >= 256) or (sub < 0):
        sub = "{:02x}".format(sub & 0xffffffff)[-2:]
        sub = int(sub, 16)
    return sub

def xor_data(self, code):
    data = self.get_data(length=self.Misc_VirtualSize)
    new_data = ""
    for b in data:
        new_data += chr(register_operation.do_xor(ord(b), code))

    self.pe.data_replace(self.PointerToRawData, new_data)

def add_data(self, code):
    data = self.get_data(length=self.Misc_VirtualSize)
    new_data = ""
    for b in data:
        new_data += chr(register_operation.do_add(ord(b), code))
```

```
  def add_last_section(self, size=512, selection_name="."+random_string(3, 7)):
        self.strip_bount_import()

        size = extend_size_alignment(size=size,
alignment=self.OPTIONAL_HEADER.SectionAlignment)

        section = SectionStructure(self.__IMAGE_SECTION_HEADER_format__,
pe=self)

        section_offset = self.sections[-1].get_file_offset() + section.sizeof()
        section.set_file_offset(section_offset)
        section.__unpack__("\x00"*section.sizeof())
        section.Name = extend_to_size(selection_name, size=8)
        section.Misc = size
        section.Misc_VirtualSize = size
        section.Misc_PhysicalAddress = size
        section.VirtualAddress =
extend_size_alignment(size=self.OPTIONAL_HEADER.SizeOfImage,
alignment=self.OPTIONAL_HEADER.SectionAlignment)
        section.SizeOfRawData = size
        section.PointerToRawData = self.sections[-1].PointerToRawData +
self.sections[-1].SizeOfRawData
        section.Characteristics = 0xE0000060
        self.__data__ = self.__data__[:section.PointerToRawData] +
"\x00"*section.SizeOfRawData + self.__data__[section.PointerToRawData:]

        self.sections.append(section)
        self.__structures__.append(section)

        self.FILE_HEADER.NumberOfSections += 1
        self.recount_SizeOfImage()

    def extend_last_section(self, size):
        size = extend_size_alignment(size=size,
alignment=self.OPTIONAL_HEADER.SectionAlignment)
        section = self.sections[-1]
```

```
        section.Misc += size
        section.Misc_VirtualSize += size
        section.Misc_PhysicalAddress += size
        section.SizeOfRawData += size
        self.data_insert(section.PointerToRawData+section.Misc_VirtualSize-size,
"\x00"*size)
        self.recount_SizeOfImage()
```