**Введение**

Привет, добро пожаловать! Я знаю что люди не любят читать введения, так что я написал как можно коротко и ясно.

Это введение в курс о разработки современных эксплойтов для Windows. Я выбрал Windows потому что Я очень хорошо знаю её внутренности и она самая популярная система. Точнее я выбрал Windows 7 SP1 x64. Хватит возиться с Windows XP, она морально устарела!

Существует много курсов по разработке эксплойтов, но они все дорогие. Если Ты финансово не можешь пройти курсы, то приходится искать разные статьи в интернете и даже смотреть видео. К сожалению, вся информация которая там преподносится не для новичков.

Это начальный курс, так что не ожидайте что тут будет легко. Разработка эскплойтов сложное и интеллектуальное ремесло. Очень сложно описать разработку простым языком. Я должен признаться что Я не эксперт, Я тоже много исследовал что бы написать эту книгу и научился многому!

В этом курсе я не только буду констатировать факты, но и покажу как прийти к ним самостоятельно. Я никогда не скажу делать что нибудь, не дав сначала техническую причину почему так. В конце курса мы будем взламывать Internet Explorer 10 и 11. Моя цель показать как перед тем как подготовить такую сложную атаку, сначала надо хорошо исследовать ее. Мы попробуем отреверсить Internet Explorer и понять самим как объекты хранятся в памяти и как мы можем использовать свои знания от этого.

Ты должен хорошо знать Х86 ассемблер. Это приказ!

Хорошо еще бы что бы ты установил две виртуальной машины с Windows 7 SP1 x64: одну с Internet Explorer 10, другую с Internet Explorer 11.

**Содержание**

1. WinDbg
2. Mona 2
3. Structure Exception Handling (SHE)
4. Heap
5. Windows Basics
6. Shellcode
7. Exploitme1 (RET EIP overwrite)
8. Exploitme2 (Stack cookies & SEH)
9. Exploitme3 (DEP)
10. Exploitme4 (ASLR)
11. Exploitme5 (Heap Spraying & UAF)
12. EMET 5.2
13. Internet Explorer 10
    1. Реверсин IE
    2. С изменения одного байта до полного захвата памяти процесса
    3. Режим бога 1
    4. Режим бога 2
    5. Использование-после-освобождение баг
14. Internet Explorer 11
    1. Часть 1
    2. Часть 2

**WinDbg**

WinDbg - хороший отладчик, но нужна практика что бы быть комфортным со всеми его командами. Я покажу только самые нужные и самые необходимые настройки. Некоторые команды мы выучим походу курса.

***Версия***

Используй 32 битную версию отладчика что бы отлаживать 32 битные приложения, также 64 версию отладчика для 64 битного приложения.

Можно менять версии отладчика между 32 и 64 одной командой:

!wow64exts.sw

***Символы***

Открой дебагер, зайди в File -> Symbol File Path и введи:

SRV\*C:\windbgsymbols\*http://msdl.microsoft.com/download/symbols

Сохраним так File -> Save Workspace

Отладчик будет использовать локальную папку как место кэша для символов. Путь (можно указать ; если путей больше одного) указывает на место где эти символы взять.

***Добавление символов во время отладки***

Что бы добавить путь к символам во время отладки введи:

.sympath+c:\symbolpath

(Команда без + изменит путь, а не добавит к нему)

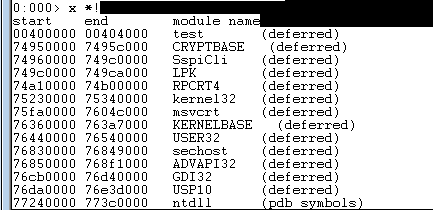
Теперь перезагрузи символы:

.reload

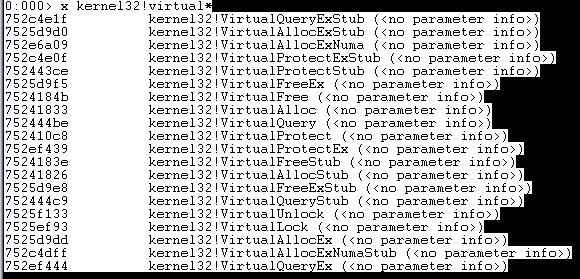
***Проверка символов***

Символы, если они есть, загружаются когда нужно. Что бы увидеть у каких модулях есть загруженные символы, введи:

x \*!



Или мы можем найти определенные названия, например virtual:



Можно заставить отладчик загрузить символы для всех модулей:

!d\*

Это только займет много времени. Зайди Debug->Break что бы остановить это.

***Локальная отладка***

Можно запустить процесс программы или подключиться к уже работающему:

1. Что бы запустить новый File -> Open Executable

2. Что бы подключиться File -> Attach to a Process

***Модули***

Когда мы производим отладку программы, мы можем перечислить загруженные модули:

!mf

Что бы перечислить специфичный модуль:

!mf m ntdll

Что бы получить заголовок модуля:

!dh ntdll

! означает что команды была вызвана с сторонней ДЛЛ, которая была внутри отладчика. Мы можем создавать свои собственные плагины что бы увеличить функциональность отладчика.

***Выражения***

Отладчик поддерживает различные выражения: мы можем написать их как они есть или вычислить его

Для примера, если EIP равно 77С6СВ70 то

bp 77c6cb71

и

bp EIP+1

эквиваленты.

Также можно использовать символы:

U ntdll!CsrSetPriorityClass+0x41

И регистры

Dd ebp+4

Числа обычно шестнадцатеричные. Что бы указать какая система используется, можно добавить префикс:

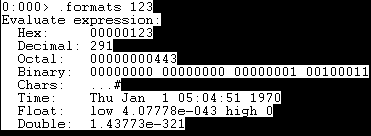
0x123 шестнадцатеричные

0n123 десятеричные

0t123 восьмеричные

0y111 двоичные

Можно использовать команду .formats что бы увидеть значения в разных представлениях:



***Регистры и псевдо-регистры***

Отладчик поддерживает несколько псевдо-регистров. Псевдо-регистры начинаются с префикса $.

Когда используются регистры или псевдо-регистры, можно добавить префикс @ который говорит говорит отладчику что дальше регистр а не символ. Если этого префикса нет, отладчик попытается интерпретировать имя как символ.

Примеры псевдо-регистров:

* $teb или @$teb (адресс ТЕВ)
* $peb или @$peb (адресс РЕВ)
* $thread или @$thread (текущий поток)

***Исключения***

Для остановки на нужном исключении, используем команду sxe.

Например когда загружен модуль остановится:

Sxe ld <module 1>, …., <module 2>

Пример:

Sx ld user32

Посмотреть список исключений

Sx

Игнорировать исключение:

Sxi ld

Она убирает эффекты первой команды.

Отладчик останавливается на single-chance исключениях а также на second-chanсe.

Как только вылетает исключение, отладчик останавливает выполнение программы и говорит что здесь имело место быть single-chance исключение.

Single-chanсe означает что исключение еще не дошло до приложения.

Когда мы восстанавливаем выполнение программы, отладчик посылает исключение приложению. Если приложение не сможет обработать исключение, отладчик снова останавливается и говорит что произошло second-chance.

Когда мы будем исследовать EMET 5.2, мы будем игнорировать single-chance single step exceptions. Что бы сделать это, команда:

Sxd sse

***Брейкпоинты***