1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 4**

1. «**Исследование подходов, применяемых при эксплуатации уязвимостей**»
2. по дисциплине «Модели безопасности компьютерных систем»
3. Выполнил
4. студент гр. 4851003/00802 Вологдин М.В.

<*подпись*>

1. Преподаватель Овасапян Т. Д.

<*подпись*>

1. Санкт-Петербург
2. 2023
3. **Цель работы**

Изучение способов эксплуатации уязвимостей и механизмов противодействия.

1. **Теоретические сведения**

**Переполнение буфера** (англ. Buffer Overflow) — явление, возникающее, когда компьютерная программа записывает данные за пределами выделенного в памяти буфера.

Переполнение буфера обычно возникает из-за неправильной работы с данными, полученными извне, и памятью, при отсутствии жесткой защиты со стороны подсистемы программирования (компилятор или интерпретатор) и операционной системы. В результате переполнения могут быть испорчены данные, расположенные следом за буфером (или перед ним).

Механизмы защиты:

**DEP**

Предотвращение выполнения данных (DEP) — это набор аппаратных и программных технологий, которые выполняют дополнительные проверки памяти для защиты от эксплойтов вредоносного кода.

С помощью аппаратного обеспечения DEP помечает все расположения памяти в процессе как неисполниемые, если расположение явно не содержит исполняемый код. Тип атак с вредоносным кодом пытается вставить и запустить код из не исполняемых расположений памяти. DEP помогает предотвратить эти атаки, перехватывая их и вызывая исключение.

**ASLR**

ASLR — это Address Space Layout Randomization, рандомизация адресного пространства. Это механизм обеспечения безопасности, который включает в себя рандомизацию виртуальных адресов памяти различных структур данных, чувствительных к атакам. Расположение в памяти целевой структуры сложно предугадать, поэтому шансы атакующего на успех малы.

Реализация ASLR в Windows тесно связана с механизмом релокации (relocation) исполняемых образов. Релокация позволяет PE-файлу загружаться не только по фиксированной предпочитаемой базе. Секция релокаций в PE-файле является ключевой структурой при перемещении образа. Она описывает, какие необходимо внести изменения в определенные элементы кода и данных для обеспечения корректного функционирования приложения по другому базовому адресу.

**Security cookie**

Глобальный cookie-файл безопасности используется для защиты от переполнения буфера в коде, скомпилированном с параметром /GS (проверка безопасности буфера), и в коде, в котором используется структурная обработка исключений. При входе в функцию с защитой от переполнения cookie-файл помещается в стек, а при выходе значение в стеке сравнивается с глобальным cookie-файлом. Любое различие между ними указывает, что произошло переполнение буфера, что приводит к немедленному завершению работы программы.

**SMEP**

SMEP(“Supervisor Mode Execution Prevention”) — предотвращение исполнения кода в режиме супервизора. Режим супервизора – это привилегированный режим работы процессора, в котором исполняется ядро ОС Windows 8. В понятиях ОС этот режим называется также режимом ядра. Противоположным ему является режим пользователя – в этом режиме исполняются пользовательские приложения.

Защита ОС строится на том, что пользовательские приложения не могут выполнять привилегированные операции, например, получить доступ к портам ввода-вывода, управляющим регистрам процессора и т.п. Кроме того, память, используемая в режиме ядра, защищена от доступа из пользовательского режима. Пользовательское приложение не может ни прочитать, ни изменить, ни выполнить код в памяти ядра напрямую. Взаимодействие с ядром ОС происходит опосредованно через интерфейс системных вызовов.

Привилегированный режим в свою очередь не имеет никаких ограничений, если бы не SMEP. Если он включен, любая попытка выполнить код, находящийся в памяти пользовательского приложения, приведет к ошибке страницы (page fault).

**SAFESEH**

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

**SEHOP**

Structured Exception Handler Overwrite Protection - начиная с Windows Vista SP1, предотвращает перезапись обработчиков, которую мы рассмотрели ранее. Является защитой на уровне ОС и не зависит от того, как вы скомпилировали свою программу.

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 забить мусором буфер и перезатереть первый обработчик, не беря во внимание все остальные. Это очень сильно усложняет эксплуатацию, так как придется подбирать правильные адреса.

1. **Ход работы**

В начале работы была установлена программа Immunity Debugger, скачан скрипт mona.py и был проведён анализ исходного файла vuln3.exe. На рисунке 1 можно увидеть список модулей, а также их параметры защиты. Заметно, что func.dll является незащищённой, имеет постоянные адреса, таким образом можно явно использовать нужные инструкции по константному адресу.

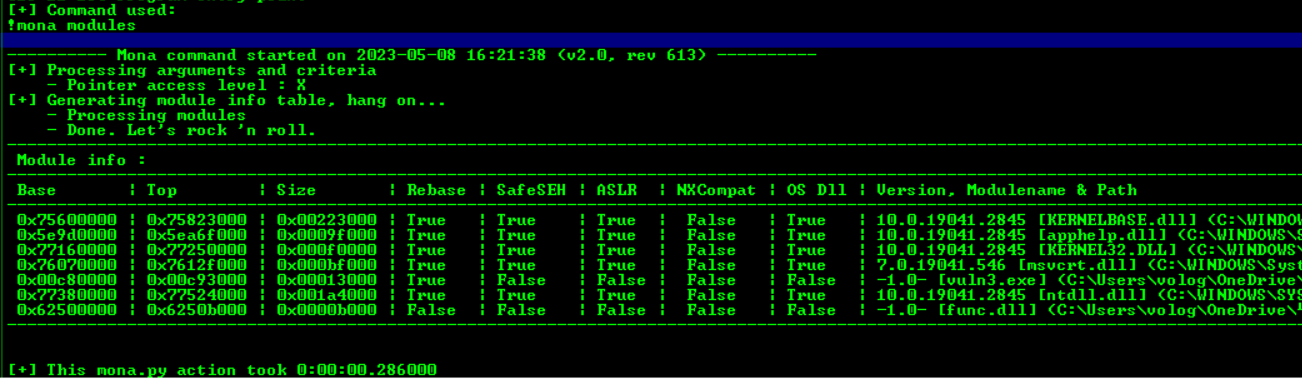


Рисунок 1 - Список модулей vuln3.exe



Рисунок 2 - Список инструкций jmp esp в библиотеке func.dll

Основная схема эксплаутации уязвимости заключается в переполнении буфера функцией strcpy, перетирание адреса возврата функции vuln\_func(), запись шеллкода на стек. Вместо адреса возврата записывается адрес инструкции jmp для исполнения шеллкода на стеке. Список возможных адресов нужных инструкций можно увидеть на рисунке 2, будем использовать первый вариант с адресом 0x62501297. После прыжка на jmp esp будет выполняться шеллкод, однако функция strcpy заканчивает копирование по первому нулевому байту. Для того, чтобы избежать проблем используется следующий метод: на стеке, как аргумент текущей функции был обнаружен адрес буфера нашего config файла, который считывается ранее функцией fread. После чего был написан мини-шеллкод (рисунок 3) из 3 команд, вычисляющий адрес буфера и прыгающий на него.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3 - Мини-шеллкод

По данному адресу шеллкод не будет обрубаться по нулевым байтам, так как fread считывает всё, включая нули. 0x14 - это смещение адреса буфера относительно esp, 0xA1D - это смещение начала шеллкода, относительно начала буфера.

Следующим шагом написан шеллкод на си. С помощью функции GetTcpTable2 считывается список портов, далее в файл записываются все порты в состоянии LISTEN. Функции достаются из библиотек с помощью хэш значений алгоритма ror13. Библиотеки подгружаются с помощью функции LoadLibraryA.

Перед сборкой в проекте были включены все оптимизации, отключены все методы защиты, а также выключена инкрементальная компоновка (рисунок 4). С помощью IDA Pro копируем в hex-формате шеллкод в config файл.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 - Настройки проекта Visual Studio

Также стоит упомянуть, что после мини-шеллкода нужно записать хотя бы 1 нулевой байт, чтобы часть шеллкода не переписала адрес буфера, который мы используем для прыжка. Таким образом config файл имеет следующую структуру: заголовочная информация + “/start” + мусорные байты для переполнения буфера + 4 байта адрес jmp esp 0x62501297 + мини-шеллкод + 4 нулевых байта + основной шеллкод.

На рисунке 5 красным выделена заголовочная информация, синим - начало мусорных байтов, на рисунке 6 красным выделен перетертый адрес возврата, синим - мини-шеллкод, черным - основной шеллкод.

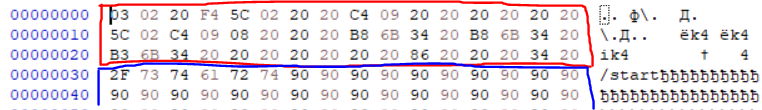


Рисунок 5 - Начало конфигурационного файла

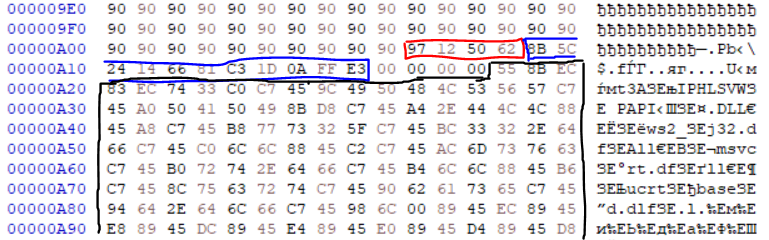


Рисунок 6 - Конец конфигурационного файла

Результат эксплуатации можно увидеть на рисунке 7.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 7 - Результат выполнения эксплуатации переполнения буфера

1. **Выводы**

В ходе выполнения лабораторной работы были изучены различные механизмы, затрудняющие эксплуатацию уязвимостей (DEP, ASLR, GS, Security Cookies, SMEP, SAFESEH, SEHOP и другие). Также был реализован шелл-код, использующий уязвимость переполнения буфера в программе vuln3.exe. В процессе работы в файл записывается список открытых портов.