ИСП РАН

Разработка и реализация метода анализа атак повторного использования кода

Алексей Вишняков

28 мая 2018 г.

ИСП РАН

Актуальность

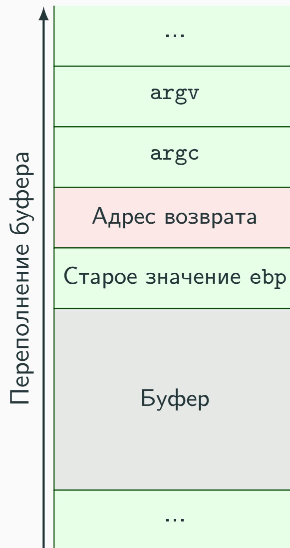
- **Уязвимость** – недостаток в системе, используя который можно злонамеренно нарушить конфиденциальность, целостность или доступность информации
- **Эксплоит** – программа, входные данные или последовательность команд, использующие уязвимость, чтобы добиться непредусмотренного поведения системы
- Эксплуатация уязвимостей может причинить колоссальный ущерб
- Крупные корпорации уделяют особое внимание анализу инцидентов информационной безопасности

Цель работы

- Для эксплуатации уязвимостей в условиях работы защитных механизмов современных операционных систем часто применяется техника *возвратно-ориентированного программирования (ROP)*
- Требуется упростить обратную инженерию ROP эксплоитов

Уязвимость переполнения буфера на стеке

- **Уязвимость переполения буфера на стеке** возникает, когда размер данных, записываемых в буфер на стеке, превышает размер этого буфера
- Переполнение буфера приводит к перезаписи ячеек выше по стеку, в т.ч. адреса возврата



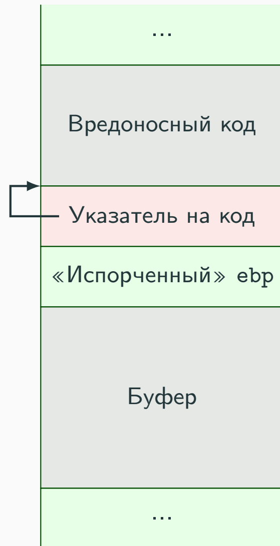
Уязвимость переполнения буфера на стеке

Выполнение произвольного кода:

- Код размещается на стеке
- Адрес возврата перезаписывается указателем на этот код

Защитный механизм:

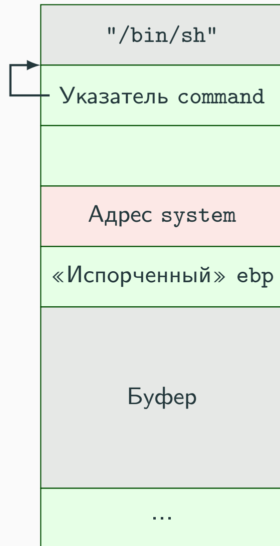
- **Ограничение исполняемых областей (DEP)** – защитный механизм операционной системы, запрещающий исполнение кода из областей памяти, помеченных как «данные»
- В частности, предотвращает выполнение вредоносного кода на стеке



Атака возврата в библиотеку

Для обхода DEP используется атака возврата в библиотеку:

- Адрес возврата подменяется адресом библиотечной функции, например, `system`
- Выше на стеке размещаются аргументы этой функции



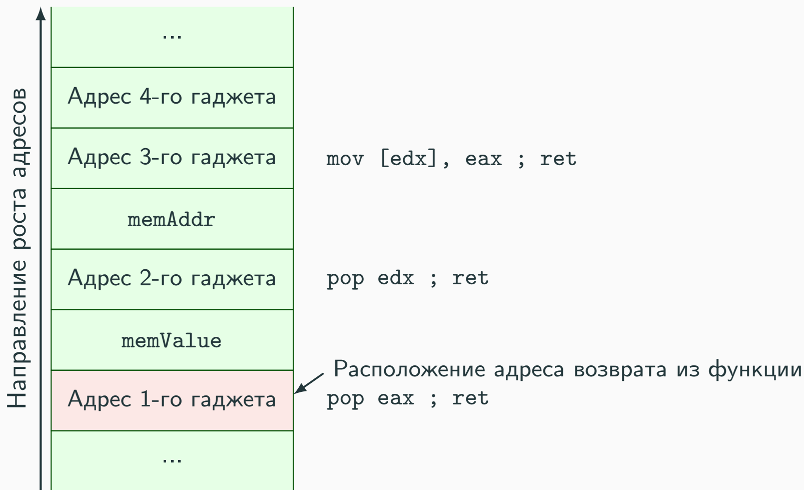
Рандомизация размещения адресного пространства

- **Рандомизация размещения адресного пространства (ASLR)** – защитный механизм операционной системы, позволяющий размещать ключевые элементы процесса (образ программы, стек, куча, динамические библиотеки) по различным адресам во время загрузки исполняемого файла
- Адрес библиотечной функции не известен до загрузки программы
- Современные реализации ASLR оставляют часть адресного пространства программы нерандомизированной:
  - В Linux адрес загрузки кода программы часто остается постоянным
  - Некоторые динамические библиотеки Windows загружаются по фиксированным адресам

Возвратно-ориентированное программирование

- **Возвратно-ориентированное программирование (ROP)** – атака повторного использования кода, позволяющая обходить DEP при наличии нерандомизированных областей памяти
- Для эксплуатации уязвимости используются кусочки кода из нерандомизированного адресного пространства программы, которые называются *гаджетами*
- Каждый гаджет выполняет некоторые вычисления (например, складывает значения двух регистров) и передает управление следующему гаджету
- Гаджеты связываются в цепочку последовательно выполняемых кусочков кода
- Таким образом, с помощью цепочки гаджетов можно выполнить некоторые вредоносные действия

Пример ROP цепочки
Запись значения memValue по адресу memAddr



Постановка задачи

Необходимо разработать и реализовать метод анализа атак повторного использования кода

Метод должен позволять по бинарной ROP цепочке:

- Восстанавливать цепочку вызванных гаджетов
- Определять семантику гаджетов
- Восстанавливать вызванные функции и их аргументы
- Выявлять системные вызовы

Определение семантики гаджетов

- *Тип гаджета* описывается семантически с помощью постусловия, булева предиката, который должен быть всегда истинным после выполнения гаджета
  - `MoveRegG: OutReg ← InReg`
  - `LoadConstG: OutReg ← [SP + Offset]`
- Набор типов гаджетов задает новую архитектуру набора команд (ISA)
- Функциональность гаджета описывается набором параметризованных типов, которым принадлежит гаджет
- Классификация гаджета выявляет набор типов и параметров, которым он соответствует

```
PUSH EAX
POP  EBX                MoveRegG: EBX ← EAX
POP  ECX                LoadConstG: ECX ← [ESP + 0]
RET
```

Пример: MongoDB Linux x86 (CVE-2013-1892)

Бинарное представление ROP цепочки

```
00000000  68 f7 16 08 07 6d 66 08  00 70 33 31 00 20 00 00  |h....mf..p31. ..|
00000010  07 00 00 00 31 00 00 00  ff ff ff ff 00 00 00 00  |....1...........|
00000020  00 00 00 00 c8 e4 16 08  00 70 33 31 00 70 33 31  |.........p31.p31|
00000030  00 00 0b 0c 00 20 00 00                           |..... ..|
00000038
```

Пример: MongoDB Linux x86 (CVE-2013-1892)

```
0x0816f768 : Asm : JMP DWORD PTR [08A1AF84h]
0x0816f768 : Call [0x8a1af84]
0x0816f768 : mmap(0x31337000, 0x2000, 0x7, 0x31, 0xffffffff, 0x0)
             from libc.so.6
0x08666d07 : Asm : ADD ESP, 00000014h ; POP EBX ; POP EBP ; RET
0x08666d07 : LoadConstG : EBX <- [ESP+20], EBP <- [ESP+24] :
             NextAddr=[ESP+28], FrameSize=32
0x08666d07 : ShiftStackG : ESP +<- 28
0x08666d07 : Values : EBX <- 0x0 ("\x00\x00\x00\x00"),
             EBP <- 0x0 ("\x00\x00\x00\x00")
0x0816e4c8 : Asm : JMP DWORD PTR [08A1AADCh]
0x0816e4c8 : Call [0x8a1aadc]
0x0816e4c8 : memcpy(0x31337000, 0xc0b0000, 0x2000) from libc.so.6
0x31337000 : Call 0x31337000
0x31337000 : Values : [ESP+4] <- 0xc0b0000, [ESP+8] <- 0x2000
```

Результаты

Были разработаны и реализованы следующие методы:

- Метод классификации гаджетов
- Метод восстановления семантики бинарной ROP цепочки
  - Производится разбиение цепочки на гаджеты
  - Гаджеты классифицируются
  - Восстанавливаются системные вызовы, функции и их агрументы

Предложенные методы были апробированы на реальных ROP эксплоитах, найденных в интернете

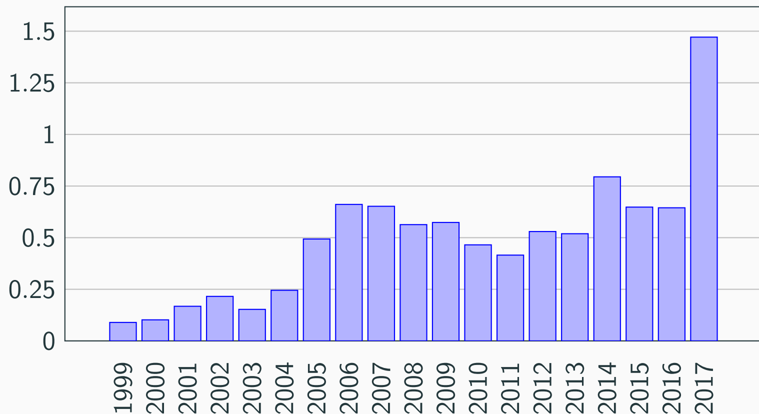
Результаты

Приложение	Номер CVE	Платформа	Гаджеты из
MongoDB	CVE-2013-1892	Linux x86	mongod
Nagios3	CVE-2012-6096	Linux x86	history.cgi
ProFTPd	CVE-2010-4221	Linux x86	proftpd
Nginx	CVE-2013-2028	Linux x64	nginx
AbsoluteFTP	CVE-2011-5164	Windows x86	MFC42.dll
ComSndFTP	N/A 2012-06-08	Windows x86	msvcrt.dll

Common Vulnerabilities and Exposures (CVE) – база данных общеизвестных уязвимостей, где каждой уязвимости присваивается идентификационный номер (CVE-год-номер), описание и хотя бы одна общедоступная ссылка

Спасибо за внимание

Актуальность



Источник: cvedetails.com/browse-by-date.php

Возвратно-ориентированное программирование

- **Гаджет** – последовательность инструкций в нерандомизированной исполняемой области памяти, которая заканчивается инструкцией передачи управления (в классическом варианте инструкцией возврата `ret`)
- Следует отметить, что архитектура x86 не требует выравнивания адресов инструкций, т.е. некоторая последовательность инструкций в программе может содержать в себе гаджет, отсутствовавший в коде программы

```
f7c7070000000f9545c3 → test edi, 0x7 ;
                       setnz BYTE PTR [ebp-0x3d]
  c7070000000f9545c3 → mov DWORD PTR [edi], 0xf000000 ;
                       xchg ebp, eax ; inc ebp ; ret
```

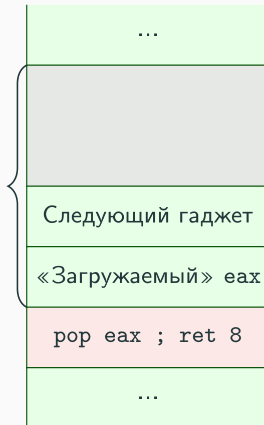
- Гаджеты собираются в цепочки, а их адреса размещаются от адреса возврата на стеке так, чтобы первый гаджет передавал управление второму, второй – третьему и т.д.

Классификация гаджетов

- Классификация гаджета выявляет набор типов и параметров, которым он соответствует
- Классификация производится на основе анализа эффектов выполнения гаджета на различных входных данных
- Инструкции гаджета транслируются в промежуточное представление
- Инструкции промежуточного представления интерпретируются с использованием теневой памяти
  - Отслеживаются обращения к регистрам и памяти
  - Начальные значения регистров и областей памяти генерируются случайным образом
  - В результате будут получены начальные и конечные значения регистров и памяти
- В результате нескольких запусков интерпретации с отличными входными данными составляется список типов и параметров с истинными постусловиями для всех запусков

Разбиение ROP цепочки на гаджеты. Фрейм гаджета

- Для декомпозиции ROP цепочки на гаджеты вводится понятие *фрейма гаджета* аналогично стековому кадру x86
- Размер фрейма
  `FrameSize = 16`
- Адрес следующего гаджета
  `NextAddr = [ESP + 4]`



Восстановление ROP цепочек

- Бинарная ROP цепочка загружается на теневой стек
- Используя информацию о фрейме гаджета, один за другим классифицируются гаджеты в цепочке
- Для восстановления значений регистров и памяти перед выполнением системного вызова или функции поддерживается общая для всех гаджетов теневая память
  - Изначально теневая память пуста
  - Производится несколько запусков интерпретации гаджета с теневой памятью в качестве начального состояния
  - Конечные значения регистров и памяти, которые не менялись от запуска к запуску, добавляются в теневую память

Восстановление функций и системных вызовов

- Имена вызванных с использованием косвенной адресации функций содержатся в таблице импортированных символов `JMP [EAX]`
  - Прототипы функций Linux берутся из man-pages
  - Прототипы функций Windows – из API Monitor
- Номер системного вызова Linux восстанавливается из значения регистра `eax` в теневой памяти
  - Прототипы системных вызовов берутся из man-pages
- Значения аргументов и строк получаются из теневой памяти

Особые случаи

- Запись на стек

```
PUSHAD
RET
```

- Запись гаджета в память

```
POP EDX ; RET // EDX = "\x89\x1c\xa8\xc3"
MOV [EAX], EDX ; RET
POP EBX ; POP EBP ; RET
JMP EAX // MOV [EAX + EBP * 4], EBX ; RET
```

- Передача управления обычному шелл-коду

```
PUSH ESP
RET
```

Дальнейшая работа

- Формальная верификация гаджетов
- Поддержка условных переходов в гаджетах
- Динамическое изменение указателя стека (Stack Pivoting)
- Гаджеты из нескольких бинарных файлов
- Разбор обычного шелл-кода