eSage Lab Лаборатория информационной безопасности

Автоматический поиск уязвимостей в программах без исходных текстов

… или введение в фаззинг

Докладчик: Олексюк Дмитрий, системный архитектор eSage Lab

Различные подходы к поиску уязвимостей

- Метод «белого ящика»

  Обычно, поиск уязвимостей по исходным текстам.

- Метод «чёрного (серого) ящика»

  Поиск уязвимостей без исходных текстов, с применением реверс-инжиниринга или без него

К чему применим фаззинг

- Сетевые протоколы
- Уязвимости при обработке файлов
- Драйверы режима ядра
- Различные интерфейсы (RPC, ActiveX компоненты)
- Многое другое

Идеальный фаззер

- Обеспечивает полное покрытие кода исследуемого приложения
- Требует для своей работы количество ресурсов, не ставящее под сомнение саму целесообразность проведения фаззинга
- Регистрирует любые аномалии в процессе исполнения исследуемого приложения
- Обеспечивает линейную масштабируемость
- *Не существует*

Этапы фаззинга

1. Анализ исследуемого приложения, разработка фаззера (опционально)
2. Генерация данных
3. Собственно, фаззинг
4. Анализ результатов

Генерация данных

- С использованием шаблонов

  Наиболее эффективный подход, но сложный и долгий в плане реализации

- На основе уже имеющихся данных

  Простота, но с сомнительной эффективностью при реализации «в лоб»

На самом деле, предпочтителен промежуточный подход

Подготовка к фаззингу

Генерация данных

- Совсем тупо: MutateGen и архивы

- Немного лучше: StgOpenStorageEx() и Visio

- Почти хорошо: Peach и HTTP протокол

Анализ покрытия кода

- Позволяет выявить, какие ветви алгоритма приложения исполнялись в процессе фаззинга

- Выбор более подходящих исходных данных для мутационного фаззинга

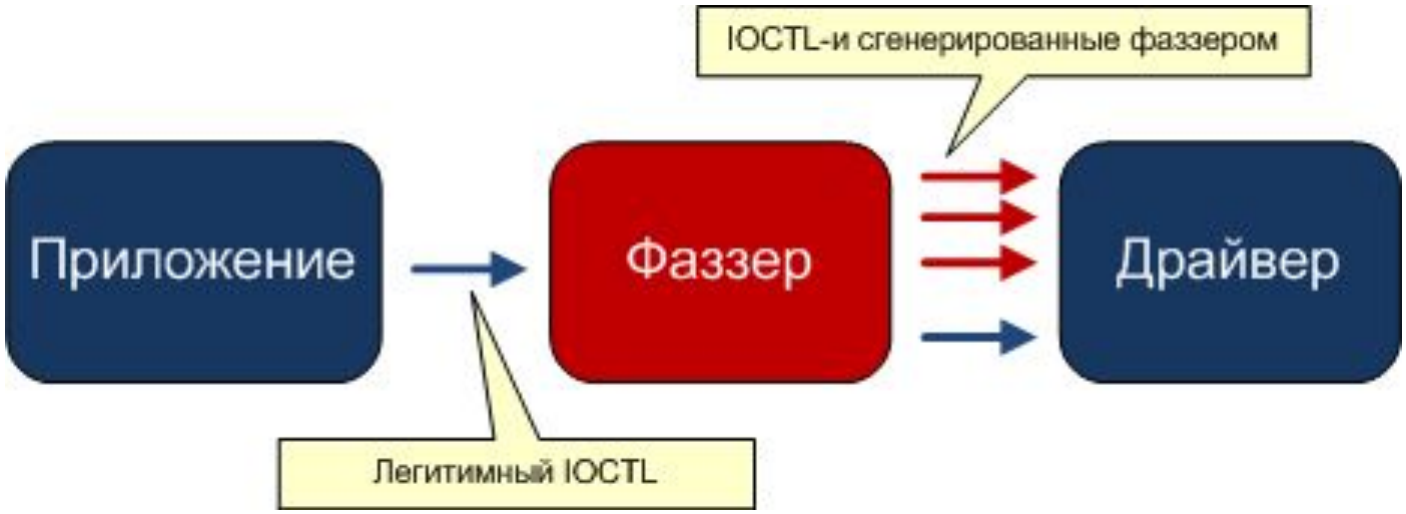
- Оценка эффективности фаззинга (чем больше покрытие кода – тем лучше)

Реализация фаззинга

Сетевые протоколы с множеством состояний?

Реализация фаззинга

IOCTL запросы к драйверам?



Анализ результатов

Мониторинг исключений

```
[+] DLL injected into the target process 8032
[+] Exit on first #AV: Yes
ModuleInit(): From process 'VISIO.EXE' (PID: 8032)
[!] EXCEPTION OCCURS:STATUS_ACCESS_VIOLATION at 0x56807161  Access
   type: Read
       Address: 0x00000102EAX=0x00000000 EBX=0x05366338
   ECX=0x00000001 EDX=0x773270b4ESI=0x0536643a EDI=0x00000000
   EBP=0x0012a244
```

Анализ результатов

Аварийные дампы

```
The stored exception information can be accessed via .ecxr.
(26dc.178c): Access violation - code c0000005 (first/second chance not available)
eax=00000700 ebx=02bf0d78 ecx=002590c4 edx=02bf0d38 esi=02bf0d38 edi=002593d8
eip=773270b4 esp=00259098 ebp=002590a8 iopl=0         nv up ei pl zr na pe nc
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for VISLIB.DLL -
VISLIB!Ordinal1+0xb18e2:
565cf57d 8b4111          mov     eax,dword ptr [ecx+11h] ds:0023:00000011=????????
ntdll!KiFastSystemCallRet:
773270b4 c3              ret
773270b5 8da42400000000  lea     esp,[esp]
773270bc 8d642400        lea     esp,[esp]
ntdll!KiIntSystemCall:
773270c0 8d542408        lea     edx,[esp+8]
773270c4 cd2e            int     2Eh
773270c6 c3              ret
773270c7 90              nop
ntdll!RtlRaiseException:
773270c8 55              push    ebp
*** Stack trace for last set context - .thread/.cxr resets it
ChildEBP RetAddr  Args to Child
WARNING: Stack unwind information not available. Following frames may be wrong.
0025a01c 565cf608 00000001 00000000 00000000 VISLIB!Ordinal1+0xb18e2
```

Анализ результатов

Отладчик и !exploitable

```
[+] Analyzing "_minidumps\0xC0000005_0x565CF57D_19.05_04.37.29.DMP"
[+] Faulting file: 000026ac.vsd
[+] Exception 0xc0000005 at address VISLIB.dll+bf57d
[+] Main module version: 12.0.6211.1000, fault module version: 12.0.6211.1000

Exploitability Classification: PROBABLY_EXPLOITABLE
Recommended Bug Title: Probably Exploitable - Data from Faulting Address controls subsequent
    Write Address starting at VISLIB!Ordinal1+0x00000000000b18e2 (Hash=0x5502736d.0x39521d59)

The data from the faulting address is later used as the target for a later write.

[+] Analyzing "_minidumps\0xC0000005_0x56807161_19.05_04.37.43.DMP"
[+] Faulting file: 0000a4e6.vsd
[+] Exception 0xc0000005 at address VISLIB.dll+2f7161
[+] Main module version: 12.0.6211.1000, fault module version: 12.0.6211.1000

*** ERROR: Module load completed but symbols could not be loaded for VISIO.EXE
Exploitability Classification: PROBABLY_NOT_EXPLOITABLE
Recommended Bug Title: Read Access Violation near NULL starting at
    VISLIB!DllCanUnloadNow+0x0000000000233971 (Hash=0x6b3b041d.0x544a027a)

This is a user mode read access violation near null, and is probably not exploitable.
```

Проблемы фаззинга

- Уязвимости в архитектуре
  Сложно предугадать, где найдёшь следующую
- В результате фаззинга не всегда происходит падение
  Особенно при pool corruption
- Не все найденные уязвимости эксплуатабельны
  А ведь уникальных воспроизводимых падений может быть сотни и тысячи!

Taint Analysis

- «Продвинутый» динамический анализ кода

- Позволяет сопоставить конкретные инструкции трассы исполнения с входными данными, которые обрабатывает исследуемая программа.

Taint Analysis и фаззинг

- Выявление участков входных данных (файла), которые не обрабатываются программой

  Сокращение количества сгенерированных данных и как следствие – времени фаззинга

- Корректировка алгоритма генерации данных в процессе фаззинга

  Увеличение покрытия кода (повышение эффективности фаззинга)

Taint Analysis и фаззинг

Возможная схема реализации

Ошибки исследователей

- Зацикленность на публичных инструментах
  - Общеизвестное не обязательно лучшее, иногда проще написать свой фаззер
- Погоня за универсальностью
  - Менее универсальный фаззер может являться более эффективным для частных случаев
- Оторванность от задач реальной жизни
  - «Крутой» и концептуальный инструмент не всегда удобен и оправдан в практическом применении

Ошибки исследователей

- Неадекватная оценка ресурсов и трудозатрат
  - Вам дороже своё рабочее время, или машинное?
- А нужен ли вам вообще фаззинг?
  - Может, проще найти уязвимость вручную?

СПАСИБО ЗА ВНИМАНИЕ

www.esagelab.ru

twitter.com/d_olex