Kernel Exploit Sample Hunting and Mining

HITBAMS2016
Amsterdam, Netherland

Fortinet

F-Secure
Introduction

- Wayne Low
  - Security Researcher @ Fortinet
  - Malware research particularly anti-HIPS techniques, providing countermeasure
  - Focusing on 0-day exploit sample discovery
  - Extremely interest into Windows exploit/vulnerability research
  - Contact: wlow (at) fortinet.com
  - Twitter: x9090

- Broderick Aquilino
  - Senior Threat Analyst @ F-Secure Labs
  - Currently working for malware protection team
  - Contact: broderick.aquilino (at) f-secure.com
  - Twitter: BrodAquilino
Agenda

Mining
- EOP vs UAC
  - Abused by malware authors
  - Differences between them
- What is WWW primitive
  - Result of mining kernel exploit sample shows classic WWW primitive kernel exploitation, eg: CVE-2013-3660 by Tavis Ormandy
- Kernel exploit sample mining
- Case study of malware families with EOP
  - Dridex/Dyre
  - Carberp/Rovnix
  - Evotob
  - Discpy

Hunting EoP anomalies
EOP vs UAC

Elevation of Privilege
- Less reliable
- Less stable
- No limitation
- Full system privilege (System integrity level)

User account control
- More reliable
- More stable
- Has limitation
- Administrator privilege (High integrity level)
What is WWW primitive

- Commonly used vector. Simple and straight forward
- Store **(write)** a specific value **(what)** to a specific kernel pointer address **(where)**, eg: HalDispatchTable
- Traditional kernel exploit uses 3 steps:
  1. Prepares a user mode buffer to store the shellcode
  2. Uses write-what-where approach to overwrite *HalDispatchTable +sizeof(void*)* with shellcode address
  3. Redirects code execution to the prepared shellcode using *NtQueryIntervalProfile*
What is WWW primitive

- Limitation:
  - Counter measures from Intel®
    - Supervisor Mode Execution Prevention (SMEP)
    - Supervisor Mode Access Prevention (SMAP)
  - Many workarounds:
    - N3phos’s exploit in CVE-2015-0058
    - Alex Ionescu’s kernel heap feng shui
- WWW primitive is prominent, but some exceptions 😞
  - CVE-2014-4113
  - CVE-2015-1701
Kernel EoP exploit sample hunting

- *NtQueryIntervalProfile & HalDispatchTable* still favorable for exploit writers 😊
- Some success stories
  - Discovery of Dridex’s CVE-2015-0057 exploit
  - Other malware families leveraging public known EOP exploits
- How to do that?
  - Windows native API calls in the process of achieving EOP
    - String search in static binary
    - String search in dynamic process memory
  - No Windows native API function name
    - Kernel exploit behavioral detection methods
Kernel EoP exploit sample hunting – WWW primitive

- Rule #1 - Generic EoP leveraging WWW
  - VT yara rule for static binary string
  - Yara rule for dynamic analysis system
  - NtQueryIntervalProfile not used by user-mode application
  - Yara rule in VT with low FP rate

```yml
rule www_kernel_exploit
{
    meta:
        description = "Typical APIs used in Write-What-Where Windows kernel exploitation"

    strings:
        $NtQueryIntervalProfile = "NtQueryIntervalProfile" nocase
        $ZwQueryIntervalProfile = "ZwQueryIntervalProfile" nocase
        $HalDispatchTable = "HalDispatchTable" nocase

    condition:
        ($NtQueryIntervalProfile or $ZwQueryIntervalProfile) and $HalDispatchTable and
        not tags contains "native"
}
```
Kernel EoP exploit sample hunting – Token Stealing

- Remember the exceptional cases without using WWW primitive?
- Upon successfully exploiting kernel vulnerability, next thing exploit will do is:
  - Elevate itself to system privilege through token stealing
    - Let’s take advantage of token stealing payload operation!
- Steps:
  - Get the EPROCESS structure of the System (process id=4) and subsequently obtains its corresponding access token address.
  - Get the EPROCESS structure of the exploit process and replace its access token address with the System’s access token.
  - As a result the exploit process possesses the same access token as the System which has the highest privilege on Windows environment.
- Used to be in ASM code... but it is not portable to other versions of Windows
- Modern exploits use documented Windows kernel API
Kernel EoP exploit sample hunting - Token Stealing (continued)

- Examples of privilege elevation payload routine taken from modern exploits
- it becomes:
  - Cleaner and portable
Kernel EoP exploit sample hunting - Token Stealing (continued)

- **Rule #2**
  - Detect token stealing operation using PsLookupProcessByProcessId and NtQuerySystemInformation
  - Specific to Win32k kernel exploit

```plaintext
rule generic_um_win32k_kernel_exploitation
{
    meta:
        description = "Typical APIs used in user-mode exploit to leverage win32k kernel mode vulnerability"

    strings:
        $PsLookupProcessByProcId = "PsLookupProcessByProcessId" ncase
        $NtQuerySystemInformation = "NtQuerySystemInformation" ncase
        $ZwQuerySystemInformation = "ZwQuerySystemInformation" ncase

    condition:
        (@NtQuerySystemInformation or @ZwQuerySystemInformation) and
        NtQuerySystemInformation && (ps.imports("user.dll") or ps.imports("user32.dll")) and
        tags contains "pe.exe" and
        not tags contains "native"

}
Kernel EoP exploit sample hunting - Token Stealing (continued)

- Rule #3
  - Detect token stealing operation using PsReferencePrimaryToken
  - Not specific to Win32k kernel exploit

```
rule generic_un_kernel_exploitation
{
    meta:
        description = "Typical APIs used in user-mode exploit to leverage kernel mode vulnerability"

    strings:
        $NtQuerySystemInformation = "NtQuerySystemInformation" nocase
        $ZwQuerySystemInformation = "ZwQuerySystemInformation" nocase
        #PsLookupProcessByProcId = "PsLookupProcessByProcessId" nocase
        #PsReferencePrimaryToken = "PsReferencePrimaryToken" nocase

    condition:
        ($NtQuerySystemInformation or $ZwQuerySystemInformation) and
        ($PsLookupProcessByProcId or #PsReferencePrimaryToken) and
        tags contains "process" and
        not tags contains "native"
}
```
Case study - Dridex

- Discovered by Rule #1
- First exploit CVE-2015-0057
  - Exploited 3 months after MS patched in Feb 2015
  - No public exploit code available that time
- Disappeared after July 2015
- Modular architecture
  - EOP exploit module downloadable from C&C as mod5

- UAC bypass module downloadable from C&C as mod4
  - Exploiting known and patched UAC vulnerability
  - Eg: AppCompat whitelisting
Case study - Discpy

- Discovered by Rule #1
- Interesting post kernel exploit payload
  - No regular token stealing
- Not a new technique but interesting idea
  - Do we really need to elevate privileges for the exploit process?
  - Other options:
    - Nullify DACL of Security Descriptor for a privileged Windows process, “Easy Local Windows Kernel Exploitation” by Cesar Cerrudo

- How about inject code to remote process from kernel mode?
  - No modification to kernel data structure
  - Kernel exploit enables code execution under kernel mode context
  - Execute APC injection routine from kernel mode
    - APC injection routine traverse active process list to find target process (eg: svchost.exe)
    - Inject APC thread to svchost.exe to run main payload
    - More stealthy
    - Bypass most of the HIPS solutions by antimalware vendors

- Update: 30 April 2016 Trend Micro discovered similar post kernel exploit payload used in Locky
Case study - Discpy

- Discpy.exe exploits CVE-2013-3660
- Transfer control to kernel mode

- Allocate kernel buffer via ExAllocatePool
- Prepares APC injector routine in kernel buffer
- Transfer code execution to kernel buffer
- Enumerate and find active svchost.exe and inject APC thread to targeted thread
- Trigger APC thread via KeInsertQueueApc that will perform final downloader/dropper routine
Hunting EoP Anomalies

- Look for unauthorized elevated processes
  - Non-system services having system integrity level
  - Processes having system integrity level with non-system Integrity level parent process
  - Processes with administrative windows privileges but < high integrity level
  - Processes Accessing Objects with Higher Integrity Level
Conclusion

- Usually means game over when reach Kernel mode
- Does not mean we have to make it easy
  - Actively hunt for them