

Kernel Exploit Sample Hunting and Mining

HITBAMS2016
Amsterdam, Netherland

FORTINET®



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

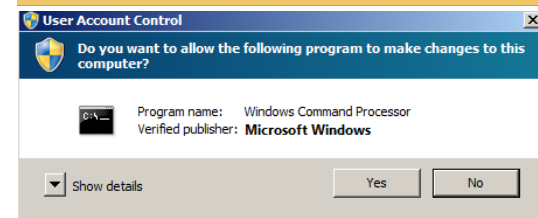
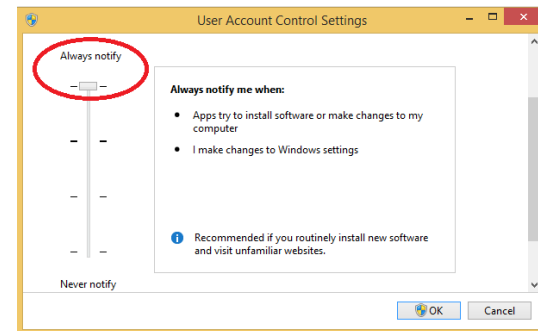
- Less reliable
- Less stable
- No limitation
- Full system privilege
(System integrity level)

User account control

- More reliable
- More stable
- Has limitation
- Administrator privilege
(High integrity level)

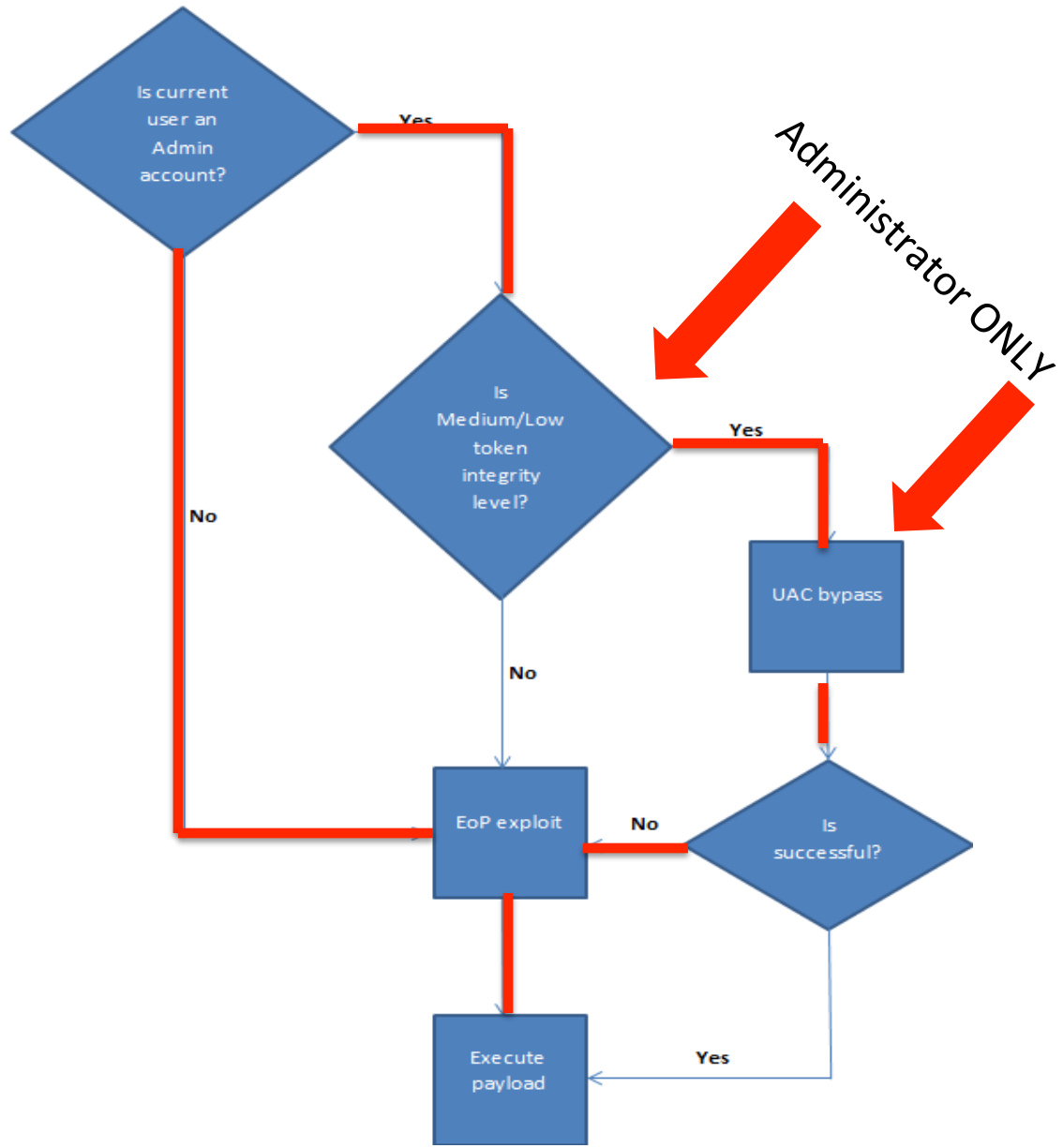
More Powerful!

explorer.exe	1840	17,452 K	27,820 K Medium
vm VMware Tray.exe	376	2,188 K	5,412 K Medium
vm VMware.exe	300	3,668 K	9,176 K Medium
cmd.exe	2668	1,672 K	2,368 K Medium
eop_explorer.exe	3192	516 K	1,940 K System
cmd.exe	3200	1,584 K	2,188 K System
cmd.exe	3208	1,572 K	1,908 K System



cmd.exe	2336	1,588 K	2,168 K High
---------	------	---------	--------------

EOP + UAC



What is WWW primitive

- Commonly used vector. Simple and straight forward
- Store (**w**rite) a specific value (**w**hat) to a specific kernel pointer address (**w**here), 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

```
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

```
int __stdcall elevate_system_privilege()  
{  
    int result;  
    PEPROCESS currentEproc;  
    PEPROCESS systemEproc;  
  
    ptrPsLookupProcessByProcessId(g_dwCurrentPid, &currentEproc);  
    ptrPsLookupProcessByProcessId(g_dwSystemPid, &systemEproc);  
    result = g_dwOffsetEprocToken;  
    *(_DWORD *)((char *)currentEproc + g_dwOffsetEprocToken) =  
    *(_DWORD *)((char *)systemEproc + g_dwOffsetEprocToken);  
    return result;  
}  
  
int elevate_privilege()  
{  
    PACCESS_TOKEN currentToken;  
    PACCESS_TOKEN SystemToken;  
    PEPROCESS currentEproc;  
  
    g_boolExploited = 1;  
    *(_DWORD *) (g_pHalDispatchTable + 4) = g_origNtQueryIntervalProfile;  
    if ( !ptrPsLookupProcessByProcessId(g_dwCurrentPid, &currentEproc) )  
    {  
        currentToken = pfnPsReferencePrimaryToken(currentEproc);  
        SystemToken = pfnPsReferencePrimaryToken(*(_DWORD *)g_PsInitialSystemProcess);  
        replace_token(currentToken, SystemToken);  
    }  
    return 0;  
}
```

Kernel EoP exploit sample hunting - Token Stealing (continued)

■ Rule #2

- Detect token stealing operation using PsLookupProcessByProcessId and NtQuerySystemInformation
- Specific to Win32k kernel exploit

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

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

    condition:
        ($NtQuerySystemInformation or $ZwQuerySystemInformation) and
        $PsLookupProcessByProcId and (pe.imports("user32.dll") or
pe.imports("gdi32.dll")) and
        tags contains "peexe" 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_um_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 "peexe" 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

```
Count of sections      4
Symbol table          00000000
Size of optional header 00E0
Linker version        10.00
Image version         0.00
Entry point           00001E06
Size of init data     00002E00
Size of image         00007000
Base of code          00001000
Image base            00400000
Section alignment     00001000
Stack                 00100000/00001000
Checksum              00000000
Machine               Intel x86
                    Wed Jul 22 21:01:15 2015
Magic optional header 0100
OS version             5.01
Subsystem version     5.01
Size of code          00001400
Size of uninit data  00000000
Size of header        00000400
Base of data          00003000
Subsystem             GUI
File alignment        00000200
Heap                  00100000/00001000
Number of dirs        16

00403050: 84 48 00 00-86 48 00 00-00 00 00-F4 49 00 00  nJ |J |
00403060: 00 00 00 00-00 00 00 00-53 00 43 00-52 00 4F 00  S C R O
00403070: 4C 00 4C 00-42 00 41 00-52 00 00-64 65 6C 6D  L L B A R delm
00403080: 65 00 00 00-6E 74 6F 73-6B 72 6E 6C-2E 65 78 65  e ntoskrnl.exe
00403090: 00 00 00 00-6E 74 64 6C-6C 2E 64 6C-6C 00 00 00  ntldr.dll
004030A0: 4E 74 51 75-65 72 79 53-79 73 74 65-6D 49 6E 66  NtQuerySystemInf
004030B0: 6F 72 6D 61-74 69 6F 6E-00 00 00-50 73 4C 6F 00 00-50 73 4C 6F 00 00-50 73 4C 6F 00 00-50 73 4C 6F
004030C0: 6F 6B 75 70-50 72 6F 63-65 73 73 42-79 50 72 6F 0kupProcessByPro
004030D0: 63 65 73 73-49 64 00 00-50 73 52 65-66 65 72 65  cessId PsRefere
004030E0: 6E 63 65 50-72 69 6D 61-72 79 54 6F-6B 65 6E 00  ncePrimaryToken
004030F0: 48 61 6C 44-69 73 70 61-74 63 68 54-61 62 6C 65  HalDispatchTable
00403100: 00 00 00 00-4E 74 51 75-65 72 79 49-6E 74 65 72  NtQueryInter
00403110: 76 61 6C 50-72 6F 66 69-6C 65 00 00-53 00 54 00  valProfile S I
00403120: 41 00 54 00-49 00 43 00-00 00 00 00-4D 00 61 00  n i t c h a
00403130: 69 00 6E 00-57 00 43 00-6C 00 61 00-73 00 73 00  i n W C l a s s
00403140: 00 00 00 00-66 69 6E 64-6D 65 00 00-66 69 6E 64  findme find
00403150: 6D 65 31 00-66 69 6E 64-6D 65 32 00-66 69 6E 64  me1 findme2 find
00403160: 6D 65 33 00-63 49 74 65-6D 73 00 00-00 00 00 00  me3 citms
```

- 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

User
Mode



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

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