Recent Exploit Trend, Mitigation and Detection Tactics

Matt Oh (Microsoft)
• DarunGrim: Patch analysis tool
• Windows Defender Advanced Threat Protection
  • Threat analysis: Vulnerabilities, APT malware
  • Sensor technology development
  • Contribution to Windows mitigation efforts: Adobe Flash Player, kernel mitigations
Objectives

00 Understand current threat landscape for Windows

01 Learn most common exploit and post-exploit methods

10 Evaluate Windows 10’s resilience to recent trend

For mitigation efforts, read Microsoft's strategy and technology improvements toward mitigating arbitrary native code execution (David Weston, Matt Miller - March 2017)
<table>
<thead>
<tr>
<th>Name</th>
<th>Affected Component</th>
<th>Exploit Type</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2016-7255</td>
<td>Win32k</td>
<td>Elevation of Privilege</td>
<td>MS16-135</td>
</tr>
<tr>
<td>CVE-2016-7256</td>
<td>Open Type Font Parser</td>
<td>Elevation of Privilege</td>
<td>MS16-132</td>
</tr>
<tr>
<td>CVE-2017-0005</td>
<td>Win32k</td>
<td>Elevation of Privilege</td>
<td>MS17-013</td>
</tr>
<tr>
<td>WannaCrypt</td>
<td>SMBv1</td>
<td>RCE</td>
<td>MS17-010</td>
</tr>
<tr>
<td>ShadowPad</td>
<td>XManager</td>
<td>Supply Chain Attack</td>
<td>Security Update Information Analysis article</td>
</tr>
<tr>
<td>CCleaner</td>
<td>CCleaner</td>
<td>Supply Chain Attack</td>
<td>Security Update Information Analysis article</td>
</tr>
</tbody>
</table>

Focus Area: Flash, Kernel, 3rd Party Applications
<table>
<thead>
<tr>
<th>Category</th>
<th>Items</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Enhancement</td>
<td>On-going code Hardening</td>
<td></td>
</tr>
<tr>
<td>Mitigation</td>
<td><strong>ASLR</strong></td>
<td><strong>Windows Vista</strong></td>
</tr>
<tr>
<td></td>
<td><strong>kASLR</strong></td>
<td><strong>Windows Vista</strong> and ongoing improvements</td>
</tr>
<tr>
<td></td>
<td><strong>CFG</strong></td>
<td><strong>Windows 10 and Windows 8.1 Update (KB3000850)</strong></td>
</tr>
<tr>
<td></td>
<td><strong>kCFG</strong></td>
<td><strong>Windows 10 Creators Update</strong></td>
</tr>
<tr>
<td></td>
<td><strong>ACG (Arbitrary Code Guard)</strong></td>
<td><strong>Microsoft Edge in the Windows 10 Creators Update</strong></td>
</tr>
<tr>
<td>Hardware-based security</td>
<td><strong>DEP</strong></td>
<td><strong>Windows XP SP2</strong></td>
</tr>
<tr>
<td></td>
<td><strong>VBS (Virtualization-based Security) – UMCI/KMCI/HVCI/HyperGuard</strong></td>
<td><strong>Windows 10</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Windows 10 S (Locked-down devices)</strong></td>
<td><strong>Windows 10 S</strong></td>
</tr>
<tr>
<td></td>
<td><strong>CET (Control-flow Enforcement Technology)</strong></td>
<td>No actual implementation exists</td>
</tr>
<tr>
<td>Policy-based mitigations</td>
<td><strong>ASR (Attack Surface Reduction) – Windows 10</strong></td>
<td>Creators Falls Update (RS3)</td>
</tr>
<tr>
<td></td>
<td><strong>Exploit Protection (WDEG) – Windows 10</strong></td>
<td>Creators Falls Update (RS3)</td>
</tr>
<tr>
<td>Security-enhanced components</td>
<td><strong>SMBv3</strong></td>
<td><strong>Windows Server 2012 and Windows 8</strong></td>
</tr>
<tr>
<td>Detection</td>
<td><strong>WDATP (Windows Defender Advanced Threat Protection)</strong></td>
<td><strong>Windows 10 Enterprise</strong></td>
</tr>
</tbody>
</table>
## Exploit Phase

<table>
<thead>
<tr>
<th>Memory Operation</th>
<th>Attack</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RW Primitives</td>
<td>Code Hardening</td>
</tr>
<tr>
<td></td>
<td>Predictable gadget locations (user/kernel)</td>
<td>ASLR/kASLR</td>
</tr>
<tr>
<td></td>
<td>Function Pointer Overwrite (call/jmp instruction hijack)</td>
<td>CFG/kCFG</td>
</tr>
<tr>
<td></td>
<td>Return Address Overwrite (ret instruction hijack)</td>
<td>CET</td>
</tr>
<tr>
<td>Control Flow Hijack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instruction Execution</td>
<td>Shellcode (Position Independent Code) on RW memory</td>
<td>DEP</td>
</tr>
<tr>
<td></td>
<td>ROP (Return-Oriented Programming)</td>
<td>Exploit Protection (WDEG)</td>
</tr>
<tr>
<td></td>
<td>Shellcode (Position Independent Code) on +X memory</td>
<td>Exploit Protection (WDEG)</td>
</tr>
<tr>
<td>+X Memory Acquisition</td>
<td>VirtualAlloc/VirtualProtect</td>
<td>ACG</td>
</tr>
<tr>
<td></td>
<td>User mode +X Memory (Kernel exploit)</td>
<td>SMEP (Supervisor Mode Execution Protection)</td>
</tr>
</tbody>
</table>
Exploit Phase
Memory Operation
### Exploit Phase

<table>
<thead>
<tr>
<th>Memory Operation</th>
<th>RW Primitives</th>
<th>Code Hardening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Flow</td>
<td>Predictable gadget locations (user/kernel)</td>
<td>ASLR/kASLR</td>
</tr>
<tr>
<td>Hijack</td>
<td>Function Pointer Overwrite (call/jmp instruction hijack)</td>
<td>CFG/kCFG</td>
</tr>
<tr>
<td></td>
<td>Return Address Overwrite (ret instruction hijack)</td>
<td>CET</td>
</tr>
<tr>
<td>Instruction</td>
<td>Shellcode (Position Independent Code) on RW memory</td>
<td>DEP</td>
</tr>
<tr>
<td>Execution</td>
<td>ROP (Return-Oriented Programming)</td>
<td>Exploit Protection (WDEG)</td>
</tr>
<tr>
<td></td>
<td>Shellcode (Position Independent Code) on +X memory</td>
<td>Exploit Protection (WDEG)</td>
</tr>
<tr>
<td>+X Memory</td>
<td>VirtualAlloc/VirtualProtect</td>
<td>ACG</td>
</tr>
<tr>
<td>Acquisition</td>
<td>User mode +X Memory (Kernel exploit)</td>
<td>SMEP (Supervisor Mode Execution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Protection)</td>
</tr>
</tbody>
</table>
• The ability to read and write target memory range
• Modern exploits heavily rely on RW Primitives for the first step of exploitation
  • Mitigations like ASLR/DEP makes non-RW Primitive exploit non-practical

RW Primitives
CVE-2015-8651: Adobe Flash Player Case

• Usage of RW Primitivies are very structured in this Adobe exploit
CVE-2015-8651: Adobe Flash Player Case

• Read Primitives

```java
private function read32x8(srcAddr:int, modeAbs:Boolean):uint {
    var _local_0:int;
    if (((isMitisSE) || (isMitisSE9))
    {
        bc[cbIndex].position = srcAddr;
        bc[cbIndex].endian = "littleEndian";
        return (bc[cbIndex].readUnsignedInt());
    }
}
```

• Write Primitives

```java
private function write32x8(destAddr:int, value:uint, modeAbs:Boolean=true):Boolean {
    if (((isMitisSE) || (isMitisSE9))
    {
        bc[cbIndex].position = destAddr;
        bc[cbIndex].endian = "littleEndian";
        return (bc[cbIndex].writeUnsignedInt(value));
    }
}
```
Windows 10 Anniversary Update: `tagWND.strName`

- Additional checks for the base and length fields, making sure that they are in the expected virtual address ranges and are not usable for RW primitives
- Exploits using this method to create an RW primitive in the kernel becomes ineffective.

```plaintext
# Child-SP    RetAddr          Call Site
00 ffff8000`65a92068 ffff800`36a5c96a nt!DbgBreakPointWithStatus
01 ffff8000`65a92070 ffff800`36a5c359 nt!KiBugCheckDebugBreak+0x12
02 ffff8000`65a920d0 ffff800`369d3094 nt!KeBugCheck2+0x8a5
03 ffff8000`65a927e0 fffdeb2`731c1fe nt!KeBugCheckEx+0x104
04 ffff8000`65a92820 fffdeb2`f71e4f96 win32kfull!DesktopVerifyHeapPointer+0x137252
05 (Inline Function) ---------`-------- win32kfull!DesktopVerifyHeapRange+0x15
06 ffff8000`65a92860 fffdeb2`f71e421b win32kfull!DesktopVerifyHeapLargeUnicodeString(struct ta
07 ffff8000`65a928a0 fffdeb2`f720c99c win32kfull!DefSetText(struct tagWND * pwnd = 0xffffded1`
08 ffff8000`65a92900 fffdeb2`f720c50a win32kfull!xxxRealDefWindowProc(struct tagWND * pwnd = 0:
09 ffff8000`65a92a80 fffdeb2`f71e51e win32kfull!xxxWrapRealDefWindowProc(struct tagWND * pwnd
```
Defenses - Remediation for RW Primitives

• Tactical mitigations
  • Structures that can be abused for RW primitives have been hunted down

• Adobe Flash Player
  • Vector, ByteArray structure hardening - Microsoft’s contribution
  • ByteArray: length checksum

• Win32k
  • tagWND.strName
Defenses - Remediation for RW Primitives

<table>
<thead>
<tr>
<th>Subroutine name</th>
<th>pdfsajoe</th>
<th>qeiofdsa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vulnerable Flash Player version</td>
<td>Below 19.0.0.185</td>
<td>19.0.0.185 and up</td>
</tr>
<tr>
<td>Mitigations</td>
<td>No latest Vector mitigations</td>
<td>Latest Vector mitigations applied</td>
</tr>
<tr>
<td>Lines of attack code</td>
<td>139 lines</td>
<td>5,021 lines</td>
</tr>
<tr>
<td>Ratio</td>
<td>1</td>
<td>36</td>
</tr>
</tbody>
</table>

*Table 1 Before and after Vector mitigation*

Exploit Phase
Control Flow Hijack
<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Operation</td>
<td>RW Primitives</td>
</tr>
<tr>
<td><strong>Control Flow Hijack</strong></td>
<td></td>
</tr>
<tr>
<td>Predictable gadget locations (user/kernel)</td>
<td></td>
</tr>
<tr>
<td>Function Pointer Overwrite (call/jmp instruction hijack)</td>
<td></td>
</tr>
<tr>
<td>Return Address Overwrite (ret instruction hijack)</td>
<td></td>
</tr>
<tr>
<td><strong>Instruction Execution</strong></td>
<td></td>
</tr>
<tr>
<td>Shellcode (Position Independent Code) on RW memory</td>
<td></td>
</tr>
<tr>
<td>ROP (Return-Oriented Programming)</td>
<td></td>
</tr>
<tr>
<td>Shellcode (Position Independent Code) on +X memory</td>
<td></td>
</tr>
<tr>
<td><strong>+X Memory Acquisition</strong></td>
<td></td>
</tr>
<tr>
<td>VirtualAlloc/VirtualProtect</td>
<td></td>
</tr>
<tr>
<td>User mode +X Memory (Kernel exploit)</td>
<td></td>
</tr>
</tbody>
</table>
Typical Exploitation Process (non-CFG target)
CVE-2016-7256 exploit: Open type font elevation of privilege

- RW Primitives
- Control Flow Hijack
- Function Pointer Overwrite
- Instruction Execution
- ROP
- +X Memory Acquisition
- User mode +X Memory (Kernel exploit)
- Instruction Execution
- Shellcode
- Kernel
- Token Swapping
Function Pointer Overwrite - CVE-2016-7256 OTF Memory Corruption

```
kd> dt fa_Callbacks 0xffffffff`0514f380
ATMFD!fa_Callbacks
+0x0000  ctx  : 0x41414141`41414141 Void
+0x0008  fread  : 0x41414141`41414141 unsigned int64  +41414141414141
+0x0010  fwrite  : 0x41414141`41414141 unsigned int64  +41414141414141
+0x0018  fseek  : 0x41414141`41414141 int  +41414141414141
+0x0020  ftell  : 0xffffffff`d28bac9d long  +fffff803d28bac9d
+0x0028  allocate  : 0x41414141`41414141 void*  +41414141414141
+0x0030  ATMfree  : 0x41414141`41414141 void  +41414141414141
+0x0038  memcmp  : 0x41414141`41414141 int  +41414141414141
+0x0040  memmove  : 0x41414141`41414141 void*  +41414141414141
+0x0048  memset  : 0x41414141`41414141 void*  +41414141414141
+0x0050  message  : 0x41414141`41414141 void  +41414141414141
+0x0058  seed  : 0x90a4f3e6`014c5f50 long  +90a4f3e6014c5f50
+0x0060  strcmp  : 0x83485e00`00000008 int  +83485e0000000008
+0x0068  strlen  : 0x8d48504e`8b4838ec int  +8d48504e8b4838ec
```

nt!qsort+0x39
Function Pointer Overwrite - CVE-2017-0005

1: kd> dt win32k!PALETTE @rax
   ...
   +0x060 pfGetNearestFromPalentry : 0x00000000`00610000 unsigned long +610000 <--
Corrupt function pointer to shellcode
• CFG bypass – non-instrumented JIT code
• ROP-less exploit
  (https://blogs.technet.microsoft.com/mmpc/2016/06/20/reverse-engineering-dubniums-flash-targeting-exploit/)
• Function pointer overwrite and CFG bypass
  • +X memory acquisition
  • Shellcode execution
• CFG/kCFG kills most of function pointer overwrite methods
Return Address Overwrite - Typical Exploitation Process (CFG target)
Return Address Overwrite

- Flash exploits in 2015, 2016 used return address overwrite mostly
- CET (Control-flow Enforcement Technology)
  - Shadow Stack

A shadow stack is a second stack for the program that is used exclusively for control transfer operations. This stack is separate from the data stack and can be enabled for operation individually in user mode or supervisor mode. When shadow stacks are enabled, the CALL instruction pushes the return address on both the data and shadow stack. The RET instruction pops the return address from both stacks and compares them. If the return addresses from the two stacks do not match, the processor signals a control protection exception (#CP). Note that the shadow stack only holds the return addresses and not parameters passed to the call instruction. See Figure 1 for an illustration of shadow stack operations on near call and ret instruction.

Exploit Phase
Instruction Execution
<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Memory Operation</strong></td>
<td></td>
</tr>
<tr>
<td>Control Flow Hijack</td>
<td></td>
</tr>
<tr>
<td>Predictable gadget locations (user/kernel)</td>
<td>ASLR/kASLR</td>
</tr>
<tr>
<td>Function Pointer Overwrite (call/jmp instruction hijack)</td>
<td>CFG/kCFG</td>
</tr>
<tr>
<td>Return Address Overwrite (ret instruction hijack)</td>
<td>CET</td>
</tr>
<tr>
<td><strong>Instruction Execution</strong></td>
<td></td>
</tr>
<tr>
<td>Shellcode (Position Independent Code) on RW memory</td>
<td>DEP</td>
</tr>
<tr>
<td>ROP (Return-Oriented Programming)</td>
<td>Exploit Protection (WDEG)</td>
</tr>
<tr>
<td>Shellcode (Position Independent Code) on +X memory</td>
<td>Exploit Protection (WDEG) ACG</td>
</tr>
<tr>
<td><strong>+X Memory Acquisition</strong></td>
<td></td>
</tr>
<tr>
<td>VirtualAlloc/VirtualProtect</td>
<td>ACG</td>
</tr>
<tr>
<td>User mode +X Memory (Kernel exploit)</td>
<td>SMEP (Supervisor Mode Execution Protection)</td>
</tr>
</tbody>
</table>
Shellcode is the traditional method of running instructions on the exploited process.

- Should maintain position independency.
- To call APIs, it relies on
  - PEB walking
    - Locate module list and find “kernel32.dll”
  - Iterate through export address table
    - Locate LoadLibrary* and GetProcAddress function calls
  - API Resolution
    - Use these two APIs to locate other APIs to use
Exploit Protection (WDEG)

• EAF (Export Address Filter)/ EAFPlus (Export Address Filter Plus)
  • Detects when code on non-module address accesses target DLL’s specific export address entries

• IAF (Import Address Filter)
  • Some advanced shellcode will use import address table to lookup LoadLibrary* and GetProcAddress APIs. This feature will detect suspicious access of the memory location.

• WDEG mitigations can be bypassed – but most popular attack methods will be caught and blocked
• Data Execution Prevention aka NX Memory killed shellcode from RW memory
• Return Oriented Programming
  • Reuse existing code snippets – gadgets
  • Turing complete
• No ultimate protection against this (not yet)
ROP - CVE-2016-7256 Kernel exploit

kd> u fffff800\'3f485b85
nt\!PpmEventTraceFailedPerfCheckStart+0x52:
  fffff800\'3f485b85 5b    pop rbx
  fffff800\'3f485b86 c3    ret

kd> u fffff800\'3f48a0e0
nt\!CcWaitForCurrentLazyWriterActivity+0xec:
  fffff800\'3f48a0e0 50    push rax
  fffff800\'3f48a0e1 5f    pop rdi
  fffff800\'3f48a0e2 c3    ret

kd> u fffff800\'3f609fc0
nt\!MmMarkPhysicalMemoryAsGood+0x54:
  fffff800\'3f609fc0 48891f    mov qword ptr [rdi],rbx
  fffff800\'3f609fc3 488b5c2430    mov rbx,qword ptr [rsp+30h]
  fffff800\'3f609fc8 4883c420    add rsp,20h
  fffff800\'3f609fcc 5f    pop rdi
  fffff800\'3f609fcd c3    ret
Defense against ROP

• CET (Control-flow Enforcement Technology)
  • Shadow Stacks
  • Indirect branch tracking

Control-flow Enforcement Technology (CET) provides the following capabilities to defend against ROP/JOP style control-flow subversion attacks:

• Shadow Stack - return address protection to defend against Return Oriented Programming,
• Indirect branch tracking - free branch protection to defend against Jump/Call Oriented Programming.

Exploit Phase
+X Memory Acquisition
## Exploit Phase

<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Memory Operation</strong></td>
<td></td>
</tr>
<tr>
<td>RW Primitives</td>
<td>Code Hardening</td>
</tr>
<tr>
<td><strong>Control Flow Hijack</strong></td>
<td></td>
</tr>
<tr>
<td>Predictable gadget locations (user/kernel)</td>
<td>ASLR/kASLR</td>
</tr>
<tr>
<td>Function Pointer Overwrite (call/jmp instruction hijack)</td>
<td>CFG/kCFG</td>
</tr>
<tr>
<td>Return Address Overwrite (ret instruction hijack)</td>
<td>CET</td>
</tr>
<tr>
<td><strong>Instruction Execution</strong></td>
<td></td>
</tr>
<tr>
<td>Shellcode (Position Independent Code) on RW memory</td>
<td>DEP</td>
</tr>
<tr>
<td>ROP (Return-Oriented Programming)</td>
<td>Exploit Protection (WDEG)</td>
</tr>
<tr>
<td>Shellcode (Position Independent Code) on +X memory</td>
<td>Exploit Protection (WDEG)</td>
</tr>
<tr>
<td><strong>+X Memory Acquisition</strong></td>
<td></td>
</tr>
<tr>
<td>VirtualAlloc/VirtualProtect</td>
<td>ACG</td>
</tr>
<tr>
<td>User mode +X Memory (Kernel exploit)</td>
<td>SMEP (Supervisor Mode Execution Protection)</td>
</tr>
</tbody>
</table>
Method for acquiring +X memory for memory corruption exploits

CVE-2015-8651 exploit example:
Defenses

• ROP detection - WDEG
  • This will detect ROP calls into suspicious APIs to VirtualProtect/VirtualAlloc
  • Supports audit/block mode

• WDATP
  • The suspicious memory allocation will be tracked and alerted

• ACG
  • +X memory allocation will trigger ACG violation if configured
  • ACG can have compatibility issue with some programs that rely on executable heap – ex. JIT
  • Supports audit/block mode
CVE-2017-0005 EoP: Using userland allocated shellcode
Modern CPUs with SMEP (Supervisor Mode Execution Protection) feature with Windows 8 and later operating system
A fatal system error has occurred.
Debugger entered on first try; Bugcheck callbacks have not been invoked.

A fatal system error has occurred.

The debuggee is ready to run
win32k!vSolidFillRect1+0x10f:
ffff960`0002794e 48894820 mov    qword ptr [rax+20h],rcx

1: kd> kp
    #    Child-SP    RetAddr Call Site
00  fffff880`070286f8  fffff803`26ff0ea nt!DbgBreakPointWithStatus
01  fffff880`07028700  fffff803`26ffe742 nt!KiBugCheckDebugBreak+0x12
02  fffff880`07028760  fffff803`26f04144 nt!KeBugCheck2+0x79f
03  fffff880`07028e80  fffff803`270726ee nt!KeBugCheckEx+0x104
04  fffff880`07028ec0  fffff803`27072674 nt! ?? ::FDODBFM::"string"+0x33594
05  fffff880`07028f00  fffff803`26f3ea09 nt! ?? ::FDODBFM::"string"+0x3351d
06  fffff880`07028f50  fffff803`26f01aee nt!MmAccessFault+0x3e9
07  fffff880`07029090  00000026`113c0000 nt!KiPageFault+0x16e ← page fault from SMEP
08  fffff880`07029228  fffff960`00052287 0x00000026`113c0000
09  fffff880`07029230  fffff960`002ac5e4 win32k!XLATEOBJ_iXlate+0x117
0a  fffff880`07029260  fffff960`0033df76 win32k!vLoadAndConvert32BitfieldsToBGRA+0x34
Post-Exploit Phase
<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable Loading</td>
<td>UMCI</td>
</tr>
<tr>
<td>CreateProcess LoadLibrary</td>
<td></td>
</tr>
<tr>
<td>Driver</td>
<td>KMCI</td>
</tr>
<tr>
<td>Code Injection</td>
<td>ACG WDATP</td>
</tr>
<tr>
<td>Thread Injection</td>
<td></td>
</tr>
<tr>
<td>Process Hollowing</td>
<td></td>
</tr>
<tr>
<td>RDI (Reflective DLL Injection)</td>
<td></td>
</tr>
<tr>
<td>APC Injection</td>
<td></td>
</tr>
<tr>
<td>Windows Feature Reuse</td>
<td>WDATP</td>
</tr>
<tr>
<td>WMI/COM/RPC</td>
<td></td>
</tr>
<tr>
<td>Powershell/VBScript/JScript</td>
<td></td>
</tr>
<tr>
<td>Kernel</td>
<td>HyperGuard, PatchGuard</td>
</tr>
<tr>
<td>Token Swapping</td>
<td></td>
</tr>
<tr>
<td>IA32_SYSENDER_EIP Hooking</td>
<td></td>
</tr>
</tbody>
</table>
Post-Exploit Phase
Code Injection
## Post-Exploit Phase

<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Executable Loading</strong></td>
<td><strong>UMCI</strong></td>
</tr>
<tr>
<td>CreateProcess LoadLibrary</td>
<td></td>
</tr>
<tr>
<td>Signed Executable Kernel Memory</td>
<td></td>
</tr>
<tr>
<td>KMCI</td>
<td></td>
</tr>
<tr>
<td><strong>Code Injection</strong></td>
<td><strong>ACG</strong></td>
</tr>
<tr>
<td>Thread Injection</td>
<td>WDATP</td>
</tr>
<tr>
<td>Process Hollowing</td>
<td></td>
</tr>
<tr>
<td>RDI (Reflective DLL Injection)</td>
<td></td>
</tr>
<tr>
<td>APC Injection</td>
<td></td>
</tr>
<tr>
<td><strong>Windows Feature Reuse</strong></td>
<td><strong>WDATP</strong></td>
</tr>
<tr>
<td>WMI/COM/RPC</td>
<td></td>
</tr>
<tr>
<td>Powershell/VBScript/JScript</td>
<td></td>
</tr>
<tr>
<td><strong>Kernel</strong></td>
<td><strong>WDATP</strong></td>
</tr>
<tr>
<td>Token Swapping</td>
<td></td>
</tr>
</tbody>
</table>
Thread Injection

- CreateProcess
- OpenProcess
- VirtualAllocEx
- WriteProcessMemory
- CreateRemoteThread

Host process

Heap

Injector

Host process

Heap

www.zeronights.org
#zeronights
Process Hollowing

Host process

Injector

CreateProcess
OpenProcess

WriteProcessMemory

ResumeThread

Entry Point

Suspend
RDI (Reflective DLL Injection)

1) Allocate and copy PE image to RWX memory

2) Resolve import addresses

3) Resolve delayed import addresses

4) Resolve relocation addresses

5) Pass control to entry point
APC Injection – WannaCrypt
DoublePulsar

**Attack machine**
- mssecsvc.exe
  - CVE-2017-0145 exploit
  - Payload package
  - Ransomware module

**Victim machine**
- lsass.exe
  - Userland shellcode
  - Reflective DLL Loading
  - Main payload launcher
- mssecsvc.exe

**Kernel**
- 1st stage shellcode
  - Install hook on srv.sys dispatch table
- Trans2 SESSION_SETUP implant (DoublePulsar)
- XOR decode
- Payload package
- Kernel shellcode

**APC Shellcode Injection**
- Drop & run

**Exploit**
- XORed payload upload

**Relation**
- Attack machine to Victim machine via mssecsvc.exe
Defense - Code Injection

• ACG can prevent any executable memory creation
• Establishing best practice policy is a challenge
  • ACG supports audit mode
  • You can collect usage of ACG over the applications and implement more solid policy with block mode
Post-Exploit Phase
Token Swapping
<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable Loading</td>
<td>UMCI</td>
</tr>
<tr>
<td>CreateProcess LoadLibrary</td>
<td>KMCI</td>
</tr>
<tr>
<td>Unsigned Executable Kernel</td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td></td>
</tr>
<tr>
<td>Code Injection</td>
<td>ACG WDATP</td>
</tr>
<tr>
<td>Thread Injection</td>
<td></td>
</tr>
<tr>
<td>Process Hollowing</td>
<td></td>
</tr>
<tr>
<td>RDI (Reflective DLL Injection)</td>
<td></td>
</tr>
<tr>
<td>APC Injection</td>
<td></td>
</tr>
<tr>
<td>Windows Feature Reuse</td>
<td>WDATP</td>
</tr>
<tr>
<td>WMI/COM/RPC</td>
<td></td>
</tr>
<tr>
<td>Poweshell/VBS/Script/JS</td>
<td></td>
</tr>
<tr>
<td>Kernel</td>
<td>WDATP</td>
</tr>
<tr>
<td>Token Swapping</td>
<td></td>
</tr>
</tbody>
</table>

Post-Exploit Phase
CVE-2016-7255 exploit: Win32k elevation of privilege

- No instruction execution involved
- Direct token modification using RW primitives
Token Swapping

```assembly
fffffa83`00a08007 90    nop
fffffa83`00a08008 e800000000 call ffffa83`00a0800d
fffffa83`00a0800d 5e    pop rsi
fffffa83`00a0800d 5e    pop rsi
fffffa83`00a0800e 4883ec38 sub rsp,38h
fffffa83`00a08012 4884e50 mov rcx,qword ptr [rsi+50h]
fffffa83`00a08016 488d542428 lea rdx,[rsp+28h]
fffffa83`00a0801b ff5658 call qword ptr [rsi+58h]
fffffa83`00a0801e 488b4e60 mov rcx,qword ptr [rsi+60h]
fffffa83`00a08022 488d542420 lea rdx,[rsp+20h]
fffffa83`00a08027 ff5658 call qword ptr [rsi+58h]
fffffa83`00a0802a 488b442420 mov rax,qword ptr [rsp+20h]
fffffa83`00a0802f 4885e668 mov r11d,dword ptr [rsi+68h]
fffffa83`00a08033 498b0c03 mov rcx,qword ptr [r11+rax]
fffffa83`00a08037 488b442428 mov rax,qword ptr [rsp+28h]
fffffa83`00a0803c 49890c03 mov qword ptr [r11+rax],rcx
fffffa83`00a08040 33c0    xor eax,eax
fffffa83`00a08042 4881c4d0020000 add rsp,200h
fffffa83`00a08049 4831db    xor rbx,rbx
fffffa83`00a0804c 4831ff    xor rdi,rdi
fffffa83`00a0804f c3    ret
```

**Call to**
PsLookupProcessByProcessId to get target EPROCESS

**Call to**
PsLookupProcessByProcessId to get SYSTEM EPROCESS

**Replace target**
EPROCESS.Token with SYSTEM’s EPROCESS.Token
Defense – WDATP Modified Token Detection

• WDATP – modified token detection

Process privilege escalation due to kernel exploit

More information about this alert
Detection source
Windows Defender ATP
Attackers typically use kernel exploits to elevate the security privileges of running processes. With elevated privileges, the affected process might be able to access sensitive files, ensure persistence, and modify system settings.
The affected process is ‘capcom_token.exe’

Recommended actions
1. Inspect the process tree of the affected process. Focus on unfamiliar processes or processes that are not digitally signed.
2. Review the machine timeline for suspicious activities, specifically those related to the affected process, that occurred right before and right after the time of the alert.
3. If the affected process is unfamiliar and is not an operating system process, submit the file for deep analysis and review detailed behavioral information from the analysis results.

Alert Process Tree
Domain Specific Issues
# Domain specific Issues

<table>
<thead>
<tr>
<th>Domain</th>
<th>Problems</th>
<th>Defenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMBv1</td>
<td>Legacy Implementation</td>
<td>SMBv3</td>
</tr>
<tr>
<td>Office</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Macro (VBA)</td>
<td>ASR (WDEG)</td>
</tr>
<tr>
<td></td>
<td>DDEAUTO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Logical Vulnerabilities</td>
<td></td>
</tr>
<tr>
<td>3rd Party Applications</td>
<td>Supply Chain Attack</td>
<td>Windows 10 S (Locked-down devices)</td>
</tr>
</tbody>
</table>
Domain Specific Issues
Office
# Domain Specific Issues

<table>
<thead>
<tr>
<th>Domain</th>
<th>Problems</th>
<th>Defenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMBv1</td>
<td>Legacy Implementation</td>
<td>SMBv3</td>
</tr>
<tr>
<td>Office</td>
<td>Macro (VBA)</td>
<td>ASR (WDEG)</td>
</tr>
<tr>
<td></td>
<td>DDEAUTO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Logical Vulnerabilities</td>
<td></td>
</tr>
<tr>
<td>3rd Party Applications</td>
<td>Supply Chain Attack</td>
<td>Device Lockdown (Windows 10 S)</td>
</tr>
</tbody>
</table>
**VBA (Visual Basic for Application)**

- It provides an application development environment
Office - VBA - Windows APIs

Private Declare PtrSafe Function VirtualAllocEx Lib "kernel32" ( _
    ByVal hProcess As Long, _
    ByVal lpAddress As Long, _
    ByVal dwSize As Long, _
    ByVal flAllocationType As Long, _
    ByVal flProtect As Long _
) As LongPtr

Private Declare PtrSafe Function VirtualFreeEx Lib "kernel32" ( _
    ByVal hProcess As Long, _
    ByVal lpAddress As Long, _
    ByVal dwSize As Long, _
    ByVal dwFreeType As Long _
) As LongPtr

Private Declare PtrSafe Function WriteProcessMemory Lib "kernel32" ( _
    ByVal hProcess As Long, _
    ByVal lpBaseAddress As LongPtr, _
    ByVal lpBuffer As Any, _
    ByVal nSize As Long, _
    ByVal lpNumberOfBytesWritten As LongPtr _
) As LongPtr

Private Declare PtrSafe Function CreateRemoteThread Lib "kernel32" ( _
    ByVal hProcess As Long, _
    ByVal lpThreadAttributes As Any, _
    ByVal dwStackSize As Long, _
    ByVal lpStartAddress As LongPtr, _
    ... _
) As LongPtr
• Ex) Retrieving Processes

Set objWMIService = GetObject("winmgmts:{impersonationLevel=impersonate}!\"" & strComputerName & "\root\cimv2")
Set Processes = objWMIService.ExecQuery("Select * from Win32_Process")

• WINDOWS MANAGEMENT INSTRUMENTATION (WMI) OFFENSE, DEFENSE, AND FORENSICS
Office - DDEAUTO

- It needs 2 dialog confirm to get code execution
- DNSMessenger
  - Pure Powershell-based threat
  - Propagated through DDEAUTO feature
Defense - Office - DDEAUTO

- For Office 2010 and later versions, to disable the DDE feature via the Registry Editor:

```
[HKEY_CURRENT_USER\Software\Microsoft\Office\version\word\Options]
DontUpdateLinks(DWORD)=1
```

- For Office 2007, to disable the DDE feature via the Registry Editor:

```
[HKEY_CURRENT_USER\Software\Microsoft\Office\12.0\word\Options\vpref]
fNoCalcLinksOnopen_00_1(DWORD)=1
```

**Defense - WDATP**

**Office/Powershell detections**

- WDATP detection time line

<table>
<thead>
<tr>
<th>Last activity</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.14.2017</td>
<td>Windows Defender AV detected an active 'Madeba' malware General</td>
</tr>
<tr>
<td>10.14.2017</td>
<td>Suspicious Powershell commandline Suspicious Activity</td>
</tr>
<tr>
<td>10.14.2017</td>
<td>Suspicious Powershell commandline Suspicious Activity</td>
</tr>
<tr>
<td>10.14.2017</td>
<td>Suspicious Powershell commandline Suspicious Activity</td>
</tr>
<tr>
<td>10.14.2017</td>
<td>Suspicious Powershell commandline Suspicious Activity</td>
</tr>
<tr>
<td>10.14.2017</td>
<td>Suspicious Powershell commandline Suspicious Activity</td>
</tr>
<tr>
<td>10.14.2017</td>
<td>An Office application ran suspicious commands Suspicious Activity</td>
</tr>
<tr>
<td>10.14.2017</td>
<td>An Office application ran suspicious commands Suspicious Activity</td>
</tr>
<tr>
<td>10.14.2017</td>
<td>Suspicious behavior by Microsoft Word was observed Document Exploit</td>
</tr>
</tbody>
</table>
Defense - WDATP
Office/Powershell detections

- Office process to suspicious Powershell command execution triggers detection
Domain Specific Issues
SMBv1
### Domain Specific Issues

<table>
<thead>
<tr>
<th>Domain</th>
<th>Problems</th>
<th>Defenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMBv1</td>
<td>Legacy Implementation</td>
<td>SMBv3</td>
</tr>
<tr>
<td>Office</td>
<td>Macro (VBA)</td>
<td>ASR (WDEG)</td>
</tr>
<tr>
<td></td>
<td>DDEAUTO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Logical Vulnerabilities</td>
<td></td>
</tr>
<tr>
<td>3rd Party Applications</td>
<td>Supply Chain Attack</td>
<td>Device Lockdown (Windows 10 S)</td>
</tr>
</tbody>
</table>
The second-stage shellcode implants *DoublePulsar* by patching the SMBv1 Transaction2 dispatch table. It overwrites one of the reserved command handlers for the *SESSION_SETUP (0xe)* subcommand of the Transaction2 request. This subcommand is reserved and not commonly used in regular code.

```
FFDFF508  add    eax, 44h ; 'H' ; packet handler shellcode start offset
FFDFF50E  mov    edi, eax
FFDFF510  lea    eax, [ebx+396h]
FFDFF516  mov    ecx, 21Ah
FFDFF518  rep movsb
FFDFF51C  pop     ebx ; eax -> packet_handler_main
FFDFF51E  mov     [ebx+00H], eax
    ; kd> dps ebx
    ; 9784f530 9787756f srv!SrvSmbOpen2
FFDFF51E  ;
    ; 9784f564 9786d5fb srv!SrvSmbCreateDirectory2
FFDFF51E  ; srv!SrvTransactionNotImplemented
FFDFF51E  ; --- Overwrite dispatch table entry for 0xe subcommand handler
FFDFF51E  ; 9784f56c 97877f2b srv!SrvTransactionNotImplemented
```
ETERNALBLUE – SMBv1 hardening and SMBv3

• SMBv1 was deprecated in 2014
• Stop using SMBv1!

• Windows 10 Fall Creators Update
  • SMBv1 is not installed by default
  • Defense-in-depth: many dispatch tables including the Transaction2 dispatch table memory area are immutable
• kCFG with VBS-enabled systems ([Windows 10 Creators Update](#))
  • Analysis of the Shadow Brokers release and mitigation with Windows 10 virtualization-based security
• ETERNALROMANCE, ETERNALBLUE would have been blocked
Domain Specific Issues

3rd Party Applications
## Domain Specific Issues

<table>
<thead>
<tr>
<th>Domain</th>
<th>Problems</th>
<th>Defenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMBv1</td>
<td>Legacy Implementation Vulnerabilities</td>
<td>SMBv3</td>
</tr>
<tr>
<td>Office</td>
<td>Macro (VBA)</td>
<td>ASR (WDEG)</td>
</tr>
<tr>
<td></td>
<td>DDEAUTO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Logical Vulnerabilities</td>
<td></td>
</tr>
<tr>
<td>3rd Party Applications</td>
<td>Supply Chain Attack</td>
<td>Device Lockdown (Windows 10 S)</td>
</tr>
</tbody>
</table>

www.zeronights.org
#zeronights
ShadowPad/CCleaner Case
ShadowPad – Stage 2

Actor (Barium)

Unknown C2 Server

Modules

Config
Plugins
Install
Online
DNS
Registry VFS
(Downloaded plugins)

Config File
Load downloaded plugins

www.zeronights.org
#zeronights
Defense – WDATP and disruptions on suspicious activities

• WDATP
  • **Operation WilySupply**: UltraEdit supply chain attack case
Defense - Windows 10 S

- Windows 10 S can provide a locked down device based upon Device Guard (UMCI/KMCI) and VBS (Virtualization Based Security)
- User’s programs will be limited to the Windows Store Apps – no 3rd party non-signed apps
  - Only Microsoft-signed and approved executables will be run on the machine
**Conclusion**

- Traditional memory corruption exploits have more challenges with Windows 10
  - RW Primitives hunted down
  - Ongoing kCFG/kASLR efforts
- Commodity malware mostly rely on social engineering
  - Office with Macro or DDEAUTO
  - Cost for producing non-social engineering exploit is too high
- Stop using SMBv1!
- Supply chain attack is a new trend to infect machines through 3\textsuperscript{rd} party application supply chain
- Windows 10 S provide VBS-enabled security environment