Who’s Watching the Watchdog? Uncovering A Privilege Escalation Vulnerability in OEM Driver

Amit Rapaport, Microsoft
> whoami

• Amit Rapaport (@realAmitRap)

• Security researcher @ THRIL Team, Windows Defender ATP

• Born & raised on Windows OS

• Deeply interested in low-level, OS internals, reverse engineering and exploitation
Talk Scope

• The story of Windows Defender ATP alert -> zero-day discovery
  • CVE-2019-5241 & CVE-2019-5242
• Demo
Where Our Story Begins

Ooops, your important files are encrypted.

If you see this text, but don't see the "Wana Decrypt0r" window, then your antivirus removed the decrypt software or you deleted it from your computer.

If you need your files you have to run the decrypt software.

Please find an application file named "@WanaDecryptor@.exe" in any folder or restore from the antivirus quarantine.

Run and follow the instructions!
WannaCry Ransomware

- Major outbreak during May 2017
- Demands $300-$600 to recover encrypted files
- Infected more than 200k machines
- Propagates mainly through SMBv1 kernel exploit - ETERNALBLUE
WannaCry Ransomware - Infection

- Machine exploitation ends up loading a lightweight, non-persistent, kernel-mode backdoor – known as DOUBLEPULSAR

- **DOUBLEPULSAR** allows an attacker to inject a custom payload into user-mode
  - Victim process is *lsass.exe*
  - Used for the initial loading of WannaCry main payload
DOUBLEPULSAR Injection Technique

Kernel

lsass.exe
DOUBLEPULSAR Injection Technique

Kernel

ZwAllocateVirtualMemory

memcpy

lsass.exe

RWX

Userland shellcode
DOUBLEPULSAR Injection Technique

**Kernel**
- ZwAllocateVirtualMemory
- memcpy
- KeInitializeApc
- KeInsertQueueApc

**lsass.exe**
- RWX
- Userland shellcode
- Victim thread
How to Detect Kernel->User APC Injection

• No hooks are allowed
• No notify callback on Mm operations
• No notify callback on APC operations...

Kernel
- ZwAllocateVirtualMemory
- memcpy
- KeInitializeApc
- KeInsertQueueApc

Isass.exe
- Userland shellcode
- RWX

Victim thread
How to Detect Kernel->User APC Injection

• We instrumented *NTOSKRNL* in Windows 10 October 2018 update to trace kernel callers doing
  • Mm operations
  • APC insert operations

• Events are traced through [Microsoft-Windows-Threat-Intelligence](http://example.com) ETW provider
Injection of potentially malicious code from the kernel

This alert is part of incident (33332)

Description

A process in kernel mode injected code into another process. This can be part of an attempt to stealthily run malicious code in the context of the target process.

The target process might exhibit behavior associated with breach activity, including attempts to open listening ports or to contact an external server.

Alert process tree

- System
  - (x)Contexon.exe
    - Kernel-initiated memory allocation in the address space services.exe
  - (x)services.exe
    - Kernel executed code in the context of services.exe
  - (x)smss.exe
Injection of potentially malicious code from the kernel

This alert is part of incident (33312)

Severity: Medium
Category: Installation
Detection source: EDR

Description
A process in kernel mode injected code into another process. This can be part of an attempt to stealthily run malicious code in the context of the target process.

The target process might exhibit behavior associated with breach activity, including attempts to open listening ports or to contact an external server.

Alert process tree

- System
  - OntokenLeex
    - Kernel-initiated memory allocation in the address space services.exe
  - .services.exe
    - Kernel executed code in the context of services.exe
  - .smss.exe
Collecting the Evidence

• Careful analysis of the alert showed that:
  • Kernel code allocated an executable region
  • Kernel code injected User APC targeting that region
  • services.exe – the only affected process on the machine

• Sounds familiar?
Hunting The Source

• Which kernel code triggered the injection?
  • Can become quite challenging...
Hunting The Source

• Which kernel code triggered the injection?
  • Can become quite challenging...

Let’s dump driver-load events on that machine:

... 

C:\Program Files\Huawei\PCManager\HwOs2Ec10x64.sys 

...
PC Manager

Security and reliability start here
Immediately optimized, PC healthier

- Hardware Check: Not checked
- Driver Management: Not checked
PC Manager

Optimizing...
Please wait while your device is optimized

Driver Management: 13 items
- Audio driver: 6.0.1.8389 - 6.0.1.8448
- Fingerprint driver: 1.1.11.29 - 1.1.11.31
- Chipset driver: 10.1.1.45
- MEI driver: 11.7.0.1057
- Independent graphics driver: 23.21.13.8892
- Graphics driver: 23.20.16.4973
- SerialIO driver: 30.100.1746.4
- DPTF driver: 8.3.10207.5567
- SGX driver: 1.9.105.41752
- WDT driver: 1.0.2.5
- Wi-Fi driver: 20.40.0.4
- BlueSoleil: 20.40.1.1
- BIOS firmware: 1.09

Security and reliability
Start here
- Diately optimized, PC healthier

Driver Management: Not checked
Hunting The Source

• Long shot, but let’s analyze these drivers import section

Looks promising!
Kernel->User Code Injector

```
\_QWORD *)Length = 409616;

ntStatus = ZwAllocateVirtualMemory(TargetProcessHandle, &UserShellcodeAddress, 0164, (PSIZE_T)Length, 0x3008u, 0x40u);
if ((ntStatus & 0x80000000) != 0)
    goto LABEL_13;
}
```
Kernel->User Code Injector

```
/* (QWORD *)Length = 4898164;

ntStatus = ZwAllocateVirtualMemory(TargetProcessHandle, &UserShellcodeAddress, 0164, (PSIZE_T)Length, 0x3000u, 0x40u);
if ( (ntstatus & 0x80000000) != 0 )
goto LABEL_13;

if ( (unsigned int)GetProcAddressFromPeb(
        TargetPebAddress,
        L"Kernel32.dll",
        (__int64)"CreateProcessW",
        &CreateProcessPtr
    ) || (unsigned int)GetProcAddressFromPeb(
        TargetPebAddress,
        L"Kernel32.dll",
        (__int64)"CloseHandle",
        &CloseHandlePtr
    ) )
{
    LABEL_12:
    ntStatus = -1073741825;
    goto LABEL_13;
}
```
Kernel->User Code Injector

User mem alloc

Resolve user functions

Copy user params
Kernel->User Code Injector

User mem alloc

Resolve user functions

Copy user params

Inject user APC
Kernel->User Code Injector

User mem alloc

Resolve user functions

Copy user params

Inject user APC

```
    "(_QWORD *)length = 409616;
    ntStatus = ZwAllocateVirtualMemory(TargetProcessHandle, &UserShellcodeAddress, 0164, (PSIZE_T)Length, 0x3000u, 0x40u);
    if ((ntstatus & 0x80000000) != 0 )
        goto LABEL_13;

    if ((unsigned int)GetExportedFunctionFromPeb(
        TargetEPProcessLocal, 
        "Kernel32.dll", 
        (unsigned int)"CreateProcessW", 
        &CreateProcessPtr)
    #if _WIN64
        // Resolve user functions
    #endif
    }

    LABEL_12:
    ntStatus = -1873741825;
    goto LABEL_13;

    memcpy_s(UserShellcodeKernelAddress, 0x800u, UserApcRoutine, (char *)nullsub_1 - (char *)UserApcRoutine);
    memset(UserShellcodeKernelAddress + 256, 0, 0x120u);
    UserShellcodeKernelAddress[256] = CreateProcessPtr;
    UserShellcodeKernelAddress[257] = CloseHandlePtr;
    memcpy_s(UserShellcodeKernelAddress + 258, 0x200u, ProcessCommandLine[1], (unsigned __int64 *)ProcessCommandLine);

    NormalRoutine = (char *)UserShellcodeAddress;
    Apc = exAllocatePool(0, 0x88u16);
    if ( Apc )
    {
        KeInitializeApc(Apc, Thread, 0164, KernelRoutine, 0164, NormalRoutine, 1, NormalRoutine + 048, AccMod);
        if ((unsigned __int8)KeInsertQueueApc((__int64)Apc, 0164, 0164, 0164 )
            return ntStatus;
```
Kernel->User Code Injector

Kernel

- ZwAllocateVirtualMemory
- memcpy
- KeInitializeApc
- KeInsertQueueApc

services.exe

Userland shellcode

Victim thread
result = (ShellcodeParamLocal->CreateProcessPtr)(
    0i64,
    &ShellcodeParamLocal->CommandLinePtr,
    0i64,
    0i64,
    v2,
    v1,
    0i64,
    0i64,
    v7,
    &v5);

if ( v6 )
    result = (ShellcodeParamLocal->CloseHandlePtr)(v6);
if ( v5 )
    result = (ShellcodeParamLocal->CloseHandlePtr)(v5);
return result;
What process gets created?

• Let’s set a breakpoint on the location where the parameter block is copied.
What process gets created?

- Let’s set a breakpoint on the location where the parameter block is copied

```
Command - Kernel \cmi\pipe\Debug\baud=115200,\pipe\reconnect\ - WinDbg:10.0.17134.12 AMD64

fff9680 49271d60 0050 0043 004d 0061 006e 0061 0067 0065
fff9680 49271d70 0072 005c 004d 0061 0074 0065 0042 006f
fff9680 49271d80 006f 006b 0053 0065 0072 0076 0069 0063
fff9680 49271d90 0065 0062 0065 0079 0065 0070 0020 002f
fff9680 49271da0 0074 0066 0072 0074 0075 0070 0030 0000

0: k> db r8
fff9680 49271d30 43 00 3a 00 5c 00 50 00 72 00 6f 00 67 00 72 00 C:\Program Files\\Huawei\\PCManager\\MateBookService.exe /startup
fff9680 49271d40 61 00 6d 00 20 00 46 00 69 00 6c 00 65 00 73 00 a a . F i l e s.
fff9680 49271d50 5c 00 48 00 75 00 61 00 77 00 65 00 69 00 5c 00 \a a . F i l e s.
fff9680 49271d60 50 00 43 00 4d 00 51 00 48 00 65 00 51 00 67 00 65 00 PC.M a n a g e.
fff9680 49271d70 72 00 5c 00 4d 00 44 00 61 00 77 00 65 00 42 00 4f 00 r . M a t e B o.
fff9680 49271d80 6f 00 6b 00 53 00 65 00 72 00 76 00 69 00 63 00 o . S e r v i c.
fff9680 49271d90 65 00 2e 00 65 00 78 00 65 00 20 00 25 00 70 00 e . e x e . / s:
fff9680 49271da0 74 00 61 00 72 00 74 00 65 00 70 00 00 00 00 t . e r t . u p:

0: k> du r8
fff9680 49271d30 "C:\Program Files\Huawei\PCManager\MateBookService.exe /startup"
fff9680 49271d90 "r\MateBookService.exe /startup"

0: k>}
```

“C:\Program Files\Huawei\PCManager\MateBookService.exe /startup”
if ( !wcsicmp(v1, L"/startup") )
{
  v2 = sub_140013340();
  if ( v2 )
  {
    // Code snippet...
  }
}

int64 sub_140013340()
{
  unsigned int v0; // ebx
  SC_HANDLE v1; // rax
  SC_HANDLE v2; // rdi
  SC_HANDLE v3; // rax
  SC_HANDLE v4; // rsi
  SC_HANDLE v5; // rcx
  struct _SERVICE_STATUS ServiceStatus; // [rsp+20h] [rbp-38h]

  v0 = 0;
  v1 = OpenSCManagerW(0I64, 0I64, 1u);
  v2 = v1;
  if ( v1 )
  {
    v3 = OpenServiceW(v1, L"MBAMainService", 0xF01FFu);
    v4 = v3;
    if ( v3 )
    {
      if ( QueryServiceStatus(v3, &ServiceStatus) && ServiceStatus.dwCurrentState == 1 && StartServiceW(v4, 0, 0I64) )
      {
      // Code snippet...
      }
    }
  }
}
Watching out for MateBookService Termination

```c
ZwQueryInformationProcess(Handle, 0i64, &ProcessPebPointer, 48i64, 0i64);
if ( ExtractCommandlineToExecute(
    &CurrentProcessCommandLine.Length,
    ProcessPebPointer,
    ProcessCommandLineToRunFromServices) >= 0 )
{
    RtlInitUnicodeString(&DestinationString, L"services.exe");
    InjectCreateProcessShellcodeToServices(&DestinationString, ProcessCommandLineToRunFromServices);
}
Watching out for MateBookService Termination

```c
ZwQueryInformationProcess(Handle, 0x164, &ProcessPebPointer, 48, 0x164);
if ( ExtractCommandLineToExecute(
    &CurrentProcessCommandLine.Length,
    ProcessPebPointer,
    ProcessCommandLineToRunFromServices) >= 0 )
{
    RtlInitUnicodeString(&DestinationString, L"services.exe");
    InjectCreateProcessShellcodeToServices(&DestinationString, ProcessCommandLineToRunFromServices);
}

v10 = ::IsProtectedProcess("(v3 + 1), &IsProtectedProcess);
if ( (v10 & 0x80000000) != 0 )
    return 0xC0000000;
if ( !IsProtectedProcess )
    return 0xC0000225;
```
Watching out for MateBookService Termination

```c
ZwQueryInformationProcess(Handle, 0i64, &ProcessPebPointer, 48i64, 0i64);
if ( ExtractCommandLineToExecute(  
    &CurrentProcessCommandLine.Length,  
    ProcessPebPointer,  
    ProcessCommandLineToRunFromServices) >= 0 )
{
    RtInitUnicodeString(&DestinationString, L"services.exe");  
    InjectCreateProcessShellcodeToServices(&DestinationString, ProcessCommandLineToRunFromServices);
}

v10 = ::IsProtectedProcess("(v3 + 1),&IsProtectedProcess);
if ( (v10 & 0x36000000) != 0 )
    return 0xC0000001;
if ( !IsProtectedProcess )
    return 0xC0000225;

KeEnterCriticalSection();
ExAcquireResourceSharedLite(&Resource, 1u);
for ( i = ProtectedProcessesAnchor; i != &ProtectedProcessesAnchor; i = *i )
{
    if ( !wcsnicmp(v3, i + 8, 0x104u) )
    {
        *v2 = 1;
        break;
    }
}
ExReleaseResourceLite(&Resource);
KeLeaveCriticalSection();
```
Quick Recap

• MateBookService.exe process terminates -> Revived by the driver

• Watched processes are held in a driver’s global list variable
I Wonder....

• If that’s a list, then there might be a way to extend it
• How does the watched processes list get extended?
I Wonder....

• If that’s a list, then there might be a way to extend it
• How does the watched processes list get extended?

-> There’s a designated IOCTL handler exactly for that purpose!
  • No validation checks on the executable directory
  • Just need to get a valid handle to the device object
Obtaining a Device Handle

• Finding #1: The device is created with DACL granting **Everyone** RW access

```c
DeviceObject = 0i64;
DeviceName.Buffer = L"\Device\Hw0s2EcDevX64";
*DeviceName.Length = 2883626;
SymbolicLinkName.Buffer = L"\DosDevices\HwOs2EcX64";
*SymbolicLinkName.Length = 3014700;
result = IoCreateDevice(a1, 8u, &DeviceName, 0x22u, 0, 0, &DeviceObject);
if ( result >= 0 )
{
    v2 = IoCreateSymbolicLink(&SymbolicLinkName, &DeviceName);
}
```
Obtaining a Device Handle

• Finding #2: the caller process is validated by its main executable path
  • Must belong to a whitelist

```c
int64 __fastcall IrpMjCreateDispatchRoutine(__int64 a1, struct IRP *Irp)
{
    struct IRP *CurrentIrp; // rdi
    PEPROCESS CurrentProcess; // rax
    NTSTATUS v4; // ebx

    Irp->IoStatus.Information = 0184;
    CurrentIrp = Irp;
    CurrentProcess = IoGetCurrentProcess();
    v4 = VerifyCallingProcessByPath(CurrentProcess);
    if ( v4 < 0 )
        v4 = -1073740767;
    CurrentIrp->IoStatus.Status = v4;
    IofCompleteRequest(CurrentIrp, 0);
    return v4;
}
```
Obtaining a Device Handle

• BUT there’s no guarantee on integrity of the caller process!
Obtaining a Device Handle

• BUT there’s no guarantee on integrity of the caller process!

• Malicious MateBookService.exe process might bypass this integrity check
Obtaining a Device Handle

• BUT there’s no guarantee on integrity of the caller process!

• Malicious MateBookService.exe process might bypass this integrity check

• Infecting our own MateBookService.exe process can be done by a low-privilege process
  • Thanks to the fact a parent process has PROCESS_ALL_ACCESS permissions over its children
IF THE WATCHDOG REVIVES ANY PROCESS I CHOOSE

COULD I ABUSE IT TO GAIN LPE?
Abusing the Watchdog to Gain LPE

Create & inject malicious code into a MateBookService.exe process

Craft special IOCTL to register LPE_POC.exe

Run LPE_POC.exe which exits immediately

The watchdog revives LPE_POC.exe as child of services.exe
Abusing the Watchdog to Gain LPE

Create & inject malicious code into a MateBookService.exe process

Craft special IOCTL to register LPE_POC.exe

Run LPE_POC.exe which exits immediately

The watchdog revives LPE_POC.exe as child of services.exe

CVE-2019-5241
DEMO TIME
Wrap up

• EDR alert -> investigation -> vulnerability find
• Reversing isn’t always the entry point
• OEM drivers – low hanging fruits for attackers
• Software devs – use OS supplied mechanisms
Acknowledgements

• A fix was released on January 9, 2019
  • Special thanks to Huawei PSIRT!
• Itai Kollmann Dekel – I wouldn’t have made it without you!
References


• https://www.huawei.com/en/psirt

• https://github.com/idan1288/ProcessHollowing32-64