

What Were They Thinking?

Anti-Virus Software Gone Wrong

May, 2006
Skywing
skywing@valhallalegends.com

Contents

1	Foreword	2
2	Introduction	3
3	Kaspersky Internet Security Suite 5.0	5
3.1	The Problems	5
3.1.1	Patching system services at runtime	5
3.1.2	Improper Validation of User-mode Pointers	7
3.1.3	Hiding Threads from User-mode	9
3.1.4	Improper Validation of Kernel Object Types	11
3.1.5	Patching non-exported, non-system-service kernel functions	17
3.1.6	Allowing User-mode Code to Access Kernel Memory . . .	22
3.2	The Solution	38
4	McAfee Internet Security Suite 2006	40
4.1	The Problem	40
4.2	The Solution	42
5	Conclusion	43

Chapter 1

Foreword

Abstract: Anti-virus software is becoming more and more prevalent on end-user computers today. Many major computer vendors (such as Dell) bundle anti-virus software and other personal security suites in the default configuration of newly-sold computer systems. As a result, it is becoming increasingly important that anti-virus software be well-designed, secure by default, and interoperable with third-party applications. Software that is installed and running by default constitutes a prime target for attack, and as such, it is especially important that said software be designed with security and interoperability in mind. In particular, this article provides examples of issues found in well-known anti-virus products. These issues range from not properly validating input from an untrusted source (especially within the context of a kernel driver) to failing to conform to API contracts when hooking or implementing an intermediary between applications and the underlying APIs upon which they rely. For popular software, or software that is installed by default, errors of this sort can become a serious problem to both system stability and security. Beyond that, it can impact the ability of independent software vendors to deploy functioning software on end-user systems.

Chapter 2

Introduction

In today's computing environment, computer security is becoming a more and more important role. The Internet poses unique dangers to networked computers, as threats such as viruses, worms, and other malicious software become more and more common.

As a result, there has been a shift towards including personal security software on most new computers sold today, such as firewall software and anti-virus software. Many new computers are operated and administered by individuals who are not experienced with the administration of a secure system. As such, they rely solely on the protection provided by a firewall or anti-virus security suite.

Given this, one would expect that firewall, anti-virus, and other personal security software would be high quality - after all, for many individuals, firewall and anti-virus software are the first (and all-too-often only) line of defense.

Unfortunately, though, most common anti-virus and personal firewall software products are full of defects that can at best make it very difficult to interoperate with (which turns out to be a serious problem for most software vendors, given how common anti-virus and firewall software is) and, at worst, compromise the very system security they advertise to protect.

This article discusses two personal security software packages that suffer from problems that are difficult to interoperate with the software and, in some cases, compromise system security. These issues are all due to shortcuts and unsafe assumptions made by the original developers.

1. Kaspersky Internet Security Suite 5.0
2. McAfee Internet Security Suite 2006

Both of these software packages include several personal security programs, including firewall and anti-virus software.

Chapter 3

Kaspersky Internet Security Suite 5.0

Kaspersky ships a personal security software suite known as Kaspersky Internet Security Suite 5.0. This package includes various personal security software programs, including a firewall and anti-virus software.

Kaspersky's anti-virus software is the primary focus of this article. Like many other anti-virus software, Kaspersky Anti-Virus provides both manual and real-time scanning capabilities.

Kaspersky's anti-virus system (KAV) employs various unsafe techniques in its kernel mode components. Some of these techniques may lead to a compromise of system security.

3.1 The Problems

3.1.1 Patching system services at runtime

Although KAV appears to use a filesystem filter, the standard Windows mechanism for intercepting accesses to files (specifically designed for applications like anti-virus software), the implementors also used a series of API-level function hooks to intercept various file accesses. Performing function hooking in kernel mode is a dangerous proposition; one must be very careful to fully validate all parameters if a function could be called from user mode (otherwise system security could be compromised by a malicious unprivileged program). Additionally, it is generally not safe to remove code hooks in kernel mode as it is difficult to prove that no threads will be running a particular code region in order to unhook

without risking bringing down the system. KAV also hooks several other system services in a misguided attempt to "protect" its processes from debuggers and process termination.

Unfortunately, the KAV programmers did not properly validate parameters passed to hooked system calls, thus leading to an opening of holes that, at the very least, allow unprivileged user mode programs to bring down the system. Some of these holes may even allow local privilege escalation (though the author has not spent the time necessary to prove whether such is possible).

KAV hooks the following system services (easily discoverable in WinDbg by comparing `nt!KeServiceDescriptorTableShadow` on a system with KAV loaded with a clean system):

```
kd> dps poi ( nt!KeServiceDescriptorTableShadow ) 1 dwo ( nt!KeServiceDescriptorTableShadow + 8 )
8191c9c8  805862de nt!NtAcceptConnectPort
8191c9cc  8056fded nt!NtAccessCheck
...
8191ca2c  f823fd00 klif!KavNtClose
...
8191ca84  f823fa20 klif!KavNtCreateProcess
8191ca88  f823fb90 klif!KavNtCreateProcessEx
8191ca8c  80647b59 nt!NtCreateProfile
8191ca90  f823fe40 klif!KavNtCreateSection
8191ca94  805747cf nt!NtCreateSemaphore
8191ca98  8059d4db nt!NtCreateSymbolicLinkObject
8191ca9c  f8240630 klif!KavNtCreateThread
8191caa0  8059a849 nt!NtCreateTimer
...
8191cbb0  f823f7b0 klif!KavNtOpenProcess
...
8191cc24  f82402f0 klif!KavNtQueryInformationFile
...
8191cc7c  f8240430 klif!KavNtQuerySystemInformation
...
8191cd00  f82405e0 klif!KavNtResumeThread
...
8191cd58  f82421f0 klif!KavNtSetInformationProcess
...
8191cdc0  f8240590 klif!KavNtSuspendThread
...
8191cdcc  f82401c0 klif!KavNtTerminateProcess
```

Additionally, KAV attempts to create several entirely new system services as a shortcut for calling kernel mode by patching the service descriptor table. This is certainly not the preferred mechanism to allow a user mode program to communicate with a driver; the programmers should have used the conventional IOCTL interface which avoids the pitfalls of patching kernel structures at runtime and having to deal with other inconveniences such as system service ordinals changing from one OS release to another.

3.1.2 Improper Validation of User-mode Pointers

Many of the hooks that KAV installs (and even the custom system services) suffer from flaws that are detrimental to the operation of the system. For instance, KAV's modified `NtOpenProcess` attempts to determine if a user address is valid by comparing it to the hardcoded value `0x7FFF0000`. On most x86 Windows systems, this address is below the highest user address (typically `0x7FFEFFFF`). However, hardcoding the size of the user address space is not a very good idea. For example, there is a boot parameter `'/3GB'` that can be set in `boot.ini` in order to change the default address space split of 2GB kernel and 2GB user to 1GB kernel and 3GB user. If a system with KAV is configured with `/3GB`, it is expected that anything that calls `NtOpenProcess` (such as the win32 `OpenProcess`) may randomly fail if parameter addresses are located above the first 2GB of the user address space:

```
.text:F82237B0 ; NTSTATUS __stdcall KavNtOpenProcess(PHANDLE ProcessHandle,
ACCESS_MASK DesiredAccess, POBJECT_ATTRIBUTES ObjectAttributes,
PCLIENT_ID ClientId)
.text:F82237B0 KavNtOpenProcess proc near          ; DATA XREF: sub_F82249D0+BFo
.
.
.
.text:F8223800    cmp     eax, 7FFF0000h ; eax = ClientId
.text:F8223805    jbe     short loc_F822380D
.text:F8223807
.text:F8223807 loc_F8223807:                      ; CODE XREF: KavNtOpenProcess+4Ej
.text:F8223807    call    ds:ExRaiseAccessViolation
```

The proper way to perform this validation would have been to use the documented `ProbeForRead` function with a SEH frame, which will automatically raise an access violation if the address is not a valid user address.

Additionally, many of KAV's custom system services do not properly validate user mode pointer arguments, which could be used to bring down the system:

```
.text:F8222BE0 ; int __stdcall KAVService10(int,PVOID OutputBuffer,int)
.text:F8222BE0 KAVService10    proc near          ; DATA XREF: .data:F8227D14o
.text:F8222BE0
.text:F8222BE0 arg_0          = dword ptr  4
.text:F8222BE0 OutputBuffer  = dword ptr  8
.text:F8222BE0 arg_8          = dword ptr  0Ch
.text:F8222BE0
.text:F8222BE0    mov     edx, [esp+OutputBuffer]
.text:F8222BE4    push    esi
.text:F8222BE5    mov     esi, [esp+4+arg_8]
.text:F8222BE9    lea     ecx, [esp+4+arg_8]
.text:F8222BED    push    ecx          ; int
.text:F8222BEE    mov     eax, [esi]    ; Unvalidated user mode pointer access
.text:F8222BF0    mov     [esp+8+arg_8], eax
.text:F8222BF4    push    eax          ; OutputBufferLength
```

```

.text:F8222BF5    mov     eax, [esp+0Ch+arg_0]
.text:F8222BF9    push    edx                ; OutputBuffer
.text:F8222BFA    push    eax                ; int
.text:F8222BFB    call    sub_F821F9A0        ; This routine internally assumes that
                                ; all pointer parameters given are valid.

.text:F8222C00    mov     edx, [esi]
.text:F8222C02    mov     ecx, [esp+4+arg_8]
.text:F8222C06    cmp     ecx, edx
.text:F8222C08    jbe     short loc_F8222C13
.text:F8222C0A    mov     eax, 0C0000173h
.text:F8222C0F    pop     esi
.text:F8222C10    retn     0Ch
.text:F8222C13    ; -----
.text:F8222C13
.text:F8222C13    loc_F8222C13:                ; CODE XREF: KAVService10+28j
.text:F8222C13    mov     [esi], ecx
.text:F8222C15    pop     esi
.text:F8222C16    retn     0Ch
.text:F8222C16    KAVService10    endp

.text:F8222C20    KAVService11    proc near                ; DATA XREF: .data:F8227D18o
.text:F8222C20
.text:F8222C20    arg_0                = dword ptr 4
.text:F8222C20    arg_4                = dword ptr 8
.text:F8222C20    arg_8                = dword ptr 0Ch
.text:F8222C20
.text:F8222C20    mov     edx, [esp+arg_4]
.text:F8222C24    push    esi
.text:F8222C25    mov     esi, [esp+4+arg_8]
.text:F8222C29    lea     ecx, [esp+4+arg_8]
.text:F8222C2D    push    ecx
.text:F8222C2E    mov     eax, [esi]        ; Unvalidated user mode pointer access
.text:F8222C30    mov     [esp+8+arg_8], eax
.text:F8222C34    push    eax
.text:F8222C35    mov     eax, [esp+0Ch+arg_0]
.text:F8222C39    push    edx
.text:F8222C3A    push    eax
.text:F8222C3B    call    sub_F8214CE0        ; This routine internally assumes
                                ; that all pointer parameters given are valid.

.text:F8222C40    test    eax, eax
.text:F8222C42    jnz     short loc_F8222C59
.text:F8222C44    mov     ecx, [esp+4+arg_8]
.text:F8222C48    mov     edx, [esi]
.text:F8222C4A    cmp     ecx, edx
.text:F8222C4C    jbe     short loc_F8222C57
.text:F8222C4E    mov     eax, STATUS_INVALID_BLOCK_LENGTH
.text:F8222C53    pop     esi
.text:F8222C54    retn     0Ch
.text:F8222C57    ; -----
.text:F8222C57
.text:F8222C57    loc_F8222C57:                ; CODE XREF: KAVService11+2Cj
.text:F8222C57    mov     [esi], ecx
.text:F8222C59
.text:F8222C59    loc_F8222C59:                ; CODE XREF: KAVService11+22j
.text:F8222C59    pop     esi
.text:F8222C5A    retn     0Ch
.text:F8222C5A    KAVService11    endp

```

3.1.3 Hiding Threads from User-mode

KAV's errors with hooking do not end with NtOpenProcess, however. One of the system services KAV hooks is NtQuerySystemInformation. This routine's behavior is modified to sometimes truncate a thread listing from certain processes when the SystemProcessesAndThreads information class is requested. This is the underlying mechanism for user mode to receive a process and thread listing of all programs running in the system, and in effect, provides a means for KAV to hide threads from user mode. The very fact that this code exists at all in KAV is curious; hiding running code from user mode is typically something that is associated with rootkits and not anti-virus software.

Aside from the potentially abusive behavior of hiding running code, this hook contains several security flaws:

1. It uses the user mode output buffer from NtQuerySystemInformation after it has been filled by the actual kernel implementation, but it does not guard against a malicious user mode program modifying this buffer or even freeing it. There is no SEH frame wrapping this function, so a user mode program could cause KAV to touch freed memory.
2. There is no validation of offsets within the returned output buffer to ensure that offsets do not refer to memory outside of the output buffer. This is problematic, because the returned data structure is actually a list of substructures that must be walked by adding an offset supplied as part of a particular substructure to the address of that substructure in order to reach the next substructure. Such an offset could be modified by user mode to actually point into kernel memory. Because the hook then sometimes writes data into what it believes is the user mode output buffer, this is an interesting avenue to explore for gaining kernel privileges from an unprivileged user mode function.

```
.text:F8224430 ; NTSTATUS __stdcall KavNtQuerySystemInformation(  
SYSTEM_INFORMATION_CLASS SystemInformationClass,  
PVOID SystemInformation,  
ULONG SystemInformationLength,  
PULONG ReturnLength)  
.text:F8224430 KavNtQuerySystemInformation proc near ; DATA XREF: sub_F82249D0+17Bo  
.text:F8224430  
.text:F8224430 var_10 = dword ptr -10h  
.text:F8224430 var_C = dword ptr -0Ch  
.text:F8224430 var_8 = dword ptr -8  
.text:F8224430 SystemInformationClass= dword ptr 4  
.text:F8224430 SystemInformation= dword ptr 8  
.text:F8224430 SystemInformationLength= dword ptr 0Ch  
.text:F8224430 ReturnLength = dword ptr 10h  
.text:F8224430 arg_24 = dword ptr 28h  
.text:F8224430  
.text:F8224430 mov eax, [esp+ReturnLength]
```

```

.text:F8224434    mov     ecx, [esp+SystemInformationLength]
.text:F8224438    mov     edx, [esp+SystemInformation]
.text:F822443C    push    ebx
.text:F822443D    push    ebp
.text:F822443E    push    esi
.text:F822443F    mov     esi, [esp+0Ch+SystemInformationClass]
.text:F8224443    push    edi
.text:F8224444    push    eax
.text:F8224445    push    ecx
.text:F8224446    push    edx
.text:F8224447    push    esi
.text:F8224448    call    OrigNtQuerySystemInformation
.text:F822444E    mov     edi, eax
.text:F8224450    cmp     esi, SystemProcessesAndThreadsInformation ;
.text:F8224450    ; Not the process / thread list API?
.text:F8224450    ; Return to caller
.text:F8224453    mov     [esp+10h+ReturnLength], edi
.text:F8224457    jnz     ret_KavNtQuerySystemInformation
.text:F822445D    xor     ebx, ebx
.text:F822445F    cmp     edi, ebx ;
.text:F822445F    ; Nothing returned?
.text:F822445F    ; Return to caller
.text:F8224461    jl      ret_KavNtQuerySystemInformation
.text:F8224467    push    ebx
.text:F8224468    push    9
.text:F822446A    push    8
.text:F822446C    call    sub_F8216730
.text:F8224471    test    al, al
.text:F8224473    jz      ret_KavNtQuerySystemInformation
.text:F8224479    mov     ebp, g_KavDriverData
.text:F822447F    mov     ecx, [ebp+0Ch]
.text:F8224482    lea     edx, [ebp+48h]
.text:F8224485    inc     ecx
.text:F8224486    mov     [ebp+0Ch], ecx
.text:F8224489    mov     ecx, ebp
.text:F822448B    call    ds:ExInterlockedPopEntrySList
.text:F8224491    mov     esi, eax
.text:F8224493    cmp     esi, ebx
.text:F8224495    jnz     short loc_F82244B7
.text:F8224497    mov     eax, [ebp+10h]
.text:F822449A    mov     ecx, [ebp+24h]
.text:F822449D    mov     edx, [ebp+1Ch]
.text:F82244A0    inc     eax
.text:F82244A1    mov     [ebp+10h], eax
.text:F82244A4    mov     eax, [ebp+20h]
.text:F82244A7    push    eax
.text:F82244A8    push    ecx
.text:F82244A9    push    edx
.text:F82244AA    call    [ebp+arg_24]
.text:F82244AD    mov     esi, eax
.text:F82244AF    cmp     esi, ebx
.text:F82244B1    jz      ret_KavNtQuerySystemInformation
.text:F82244B7
.text:F82244B7    loc_F82244B7: ; CODE XREF: KavNtQuerySystemInformation+65j
.text:F82244B7    mov     edi, [esp+10h+SystemInformation]
.text:F82244BB    mov     dword ptr [esi], 8
.text:F82244C1    mov     dword ptr [esi+4], 9

```

```

.text:F82244C8    mov     [esi+8], ebx
.text:F82244CB    mov     [esi+34h], ebx
.text:F82244CE    mov     dword ptr [esi+3Ch], 1
.text:F82244D5    mov     [esi+10h], bl
.text:F82244D8    mov     [esi+30h], ebx
.text:F82244DB    mov     [esi+0Ch], ebx
.text:F82244DE    mov     [esi+38h], ebx
.text:F82244E1    mov     ebp, 13h
.text:F82244E6
.text:F82244E6 LoopThreadProcesses:      ; CODE XREF: KavNtQuerySystemInformation+ECj
.text:F82244E6    mov     dword ptr [esi+40h], 4 ;
.text:F82244E6    ; Loop through the returned list of processes and threads.
.text:F82244E6    ; For each process, we shall check to see if it is a
.text:F82244E6    ; special (protected) process. If so, then we might
.text:F82244E6    ; decide to remove its threads from the listing returned
.text:F82244E6    ; by setting the thread count to zero.
.text:F82244ED    mov     [esi+48h], ebx
.text:F82244F0    mov     [esi+44h], ebp
.text:F82244F3    mov     eax, [edi+SYSTEM_PROCESSES.ProcessId]
.text:F82244F6    push    ebx
.text:F82244F7    push    esi
.text:F82244F8    mov     [esi+4Ch], eax
.text:F82244FB    call    KavCheckProcess
.text:F8224500    cmp     eax, 7
.text:F8224503    jz      short CheckNextThreadProcess
.text:F8224505    cmp     eax, 1
.text:F8224508    jz      short CheckNextThreadProcess
.text:F822450A    cmp     eax, ebx
.text:F822450C    jz      short CheckNextThreadProcess
.text:F822450E    mov     [edi+SYSTEM_PROCESSES.ThreadCount], ebx ;
                    ; Zero thread count out (hide process threads)
.text:F8224511
.text:F8224511 CheckNextThreadProcess:      ; CODE XREF: KavNtQuerySystemInformation+D3j
.text:F8224511    ; KavNtQuerySystemInformation+D8j ...
.text:F8224511    mov     eax, [edi+SYSTEM_PROCESSES.NextEntryDelta]
.text:F8224513    cmp     eax, ebx
.text:F8224515    setz    cl
.text:F8224518    add     edi, eax
.text:F822451A    cmp     cl, bl
.text:F822451C    jz      short LoopThreadProcesses

```

3.1.4 Improper Validation of Kernel Object Types

Windows exposes many kernel features through a series of “kernel objects”. These objects may be acted upon by user mode through the user of handles. Handles are integral values that are translated by the kernel into pointers to a particular object upon which something (typically a system service) interacts with on behalf of a caller. All objects share the same handle namespace.

Because of this handle namespace sharing between objects of different types, one of the jobs of a system service inspecting a handle is to verify that the object that it refers to is of the expected type. This is accomplished by an object manager routine `ObReferenceObjectByHandle`, which performs the translation of handles

to object pointers and does an optional built-in type check by comparing a type field in the standard object header to a passed in type.

Since KAV hooks system services, it inevitably must deal with kernel handles. Unfortunately, it does not do so correctly. In some cases, it does not ensure that a handle refers to an object of a particular type before using the object pointer. This will result in corruption or a system crash if a handle of the wrong type is passed to a system service.

One such case is the KAV NtResumeThread hook, which attempts to track the state of running threads in the system. In this particular case, it does not seem possible for user mode to crash the system by passing an object of the wrong type as the returned object pointer because it is simply used as a key in a lookup table that is prepopulated with thread object pointers. KAV also hooks NtSuspendThread for similar purposes, and this hook has the same problem with the validation of object handle types.

```
.text:F82245E0 ; NTSTATUS __stdcall KavNtResumeThread(
HANDLE ThreadHandle,
PULONG PreviousSuspendCount)
.text:F82245E0 KavNtResumeThread proc near          ; DATA XREF: sub_F82249D0+FB0
.text:F82245E0
.text:F82245E0 ThreadHandle = dword ptr 8
.text:F82245E0 PreviousSuspendCount= dword ptr 0Ch
.text:F82245E0
.text:F82245E0      push     esi
.text:F82245E1      mov      esi, [esp+ThreadHandle]
.text:F82245E5      test     esi, esi
.text:F82245E7      jz       short loc_F8224620
.text:F82245E9      lea      eax, [esp+ThreadHandle] ;
.text:F82245E9      ; This should pass an object type here!
.text:F82245ED      push     0 ; HandleInformation
.text:F82245EF      push     eax ; Object
.text:F82245F0      push     0 ; AccessMode
.text:F82245F2      push     0 ; ObjectType
.text:F82245F4      push     0F0000h ; DesiredAccess
.text:F82245F9      push     esi ; Handle
.text:F82245FA      mov      [esp+18h+ThreadHandle], 0
.text:F8224602      call     ds:ObReferenceObjectByHandle
.text:F8224608      test     eax, eax
.text:F822460A      jl       short loc_F8224620
.text:F822460C      mov      ecx, [esp+ThreadHandle]
.text:F8224610      push     ecx
.text:F8224611      call     KavUpdateThreadRunningState
.text:F8224616      mov      ecx, [esp+ThreadHandle] ; Object
.text:F822461A      call     ds:ObfDereferenceObject
.text:F8224620
.text:F8224620 loc_F8224620: ; CODE XREF: KavNtResumeThread+7j
.text:F8224620      ; KavNtResumeThread+2Aj
.text:F8224620      mov      edx, [esp+PreviousSuspendCount]
.text:F8224624      push     edx
.text:F8224625      push     esi
.text:F8224626      call     OrigNtResumeThread
.text:F822462C      pop      esi
```

```

.text:F822462D    retn     8
.text:F822462D KavNtResumeThread endp
.text:F822462D

.text:F8224590 ; NTSTATUS __stdcall KavNtSuspendThread(
HANDLE ThreadHandle,
PULONG PreviousSuspendCount)
.text:F8224590 sub_F8224590    proc near                ; DATA XREF: sub_F82249D0+113o
.text:F8224590
.text:F8224590 ThreadHandle    = dword ptr 8
.text:F8224590 PreviousSuspendCount= dword ptr 0Ch
.text:F8224590
.text:F8224590    push     esi
.text:F8224591    mov      esi, [esp+ThreadHandle]
.text:F8224595    test     esi, esi
.text:F8224597    jz       short loc_F82245D0
.text:F8224599    lea      eax, [esp+ThreadHandle] ;
.text:F8224599                                ; This should pass an object type here!
.text:F822459D    push     0                ; HandleInformation
.text:F822459F    push     eax                ; Object
.text:F82245A0    push     0                ; AccessMode
.text:F82245A2    push     0                ; ObjectType
.text:F82245A4    push     0F0000h          ; DesiredAccess
.text:F82245A9    push     esi                ; Handle
.text:F82245AA    mov      [esp+18h+ThreadHandle], 0
.text:F82245B2    call     ds:ObReferenceObjectByHandle
.text:F82245B8    test     eax, eax
.text:F82245BA    jnl      short loc_F82245D0
.text:F82245BC    mov      ecx, [esp+ThreadHandle]
.text:F82245C0    push     ecx
.text:F82245C1    call     KavUpdateThreadSuspendedState
.text:F82245C6    mov      ecx, [esp+ThreadHandle] ; Object
.text:F82245CA    call     ds:ObfDereferenceObject
.text:F82245D0
.text:F82245D0 loc_F82245D0:                ; CODE XREF: sub_F8224590+7j
.text:F82245D0                                ; sub_F8224590+2Aj
.text:F82245D0    mov      edx, [esp+PreviousSuspendCount]
.text:F82245D4    push     edx
.text:F82245D5    push     esi
.text:F82245D6    call     OrigNtSuspendThread
.text:F82245DC    pop      esi
.text:F82245DD    retn     8
.text:F82245DD sub_F8224590    endp
.text:F82245DD

```

Not all of KAV's hooks are so fortunate, however. The NtTerminateProcess hook that KAV installs looks into the body of the object referred to by the process handle parameter of the function in order to determine the name of the process being terminated. However, KAV fails to validate that the object handle given by user mode really refers to a process object.

This is unsafe for several reasons, which may be well known to the reader if one is experienced with Windows kernel programming.

1. The kernel process structure definition (EPROCESS) changes frequently from OS release to OS release, and even between service packs. As a result, it is not generally safe to access this structure directly.
2. Because KAV does not perform proper type checking, it is possible to pass an object handle to a different kernel object - say, a mutex - which may cause KAV to bring down the system because the internal object structures of a mutex (or any other kernel object) are not compatible with that of a process object.

KAV attempts to work around the first problem by attempting to discover the offset of the member in the EPROCESS structure that contains the process name at runtime. The algorithm used is to scan forward one byte at a time from the start of the process object pointer until a sequence of bytes identifying the name of the initial system process is discovered. (This routine is called in the context of the initial system process). This routine appears to be very common amongst anti-virus and other low-level products that attempt to make use of the image file name associated with a process.

```
.text:F82209E0 KavFindEprocessNameOffset proc near      ; CODE XREF: sub_F8217A60+FCp
.text:F82209E0     push     ebx
.text:F82209E1     push     esi
.text:F82209E2     push     edi
.text:F82209E3     call     ds:IoGetCurrentProcess
.text:F82209E9     mov      edi, ds:strncmp
.text:F82209EF     mov      ebx, eax
.text:F82209F1     xor      esi, esi
.text:F82209F3
.text:F82209F3 loc_F82209F3:                          ; CODE XREF: KavFindEprocessNameOffset+2Ej
.text:F82209F3     lea      eax, [esi+ebx]
.text:F82209F6     push     6                          ; size_t
.text:F82209F8     push     eax                          ; char *
.text:F82209F9     push     offset aSystem ; "System"
.text:F82209FE     call     edi ; strcmp
.text:F8220A00     add      esp, 0Ch
.text:F8220A03     test     eax, eax
.text:F8220A05     jz       short loc_F8220A16
.text:F8220A07     inc      esi
.text:F8220A08     cmp      esi, 3000h
.text:F8220A0E     jl       short loc_F82209F3
.text:F8220A10     pop      edi
.text:F8220A11     pop      esi
.text:F8220A12     xor      eax, eax
.text:F8220A14     pop      ebx
.text:F8220A15     retn
.text:F8220A16 ; -----
.text:F8220A16
.text:F8220A16 loc_F8220A16:                          ; CODE XREF: KavFindEprocessNameOffset+25j
.text:F8220A16     mov      eax, esi
.text:F8220A18     pop      edi
.text:F8220A19     pop      esi
.text:F8220A1A     pop      ebx
```

```

.text:F8220A1B    retn
.text:F8220A1B KavFindEprocessNameOffset endp

.text:F8217B5C    call    KavFindEprocessNameOffset
.text:F8217B61    mov     g_EprocessNameOffset, eax

```

Given a handle to an object of the wrong type, KAV will read from the returned object body pointer in an attempt to determine the name of the process being destroyed. This will typically run off the end of the structure for an object that is not a process object (the Process object is very large compared to some objects, such as a Mutex object, and the offset of the process name within this structure is typically several hundred bytes or more). It is expected that this will cause the system to crash if a bad handle is passed to NtTerminateProcess.

```

.text:F82241C0 ; NTSTATUS __stdcall KavNtTerminateProcess(HANDLE ThreadHandle,NTSTATUS ExitStatus)
.text:F82241C0 KavNtTerminateProcess proc near          ; DATA XREF: sub_F82249D0+AB0
.text:F82241C0
.text:F82241C0 var_58             = dword ptr -58h
.text:F82241C0 ProcessObject      = dword ptr -54h
.text:F82241C0 ProcessData       = KAV_TERMINATE_PROCESS_DATA ptr -50h
.text:F82241C0 var_4             = dword ptr -4
.text:F82241C0 ProcessHandle     = dword ptr 4
.text:F82241C0 ExitStatus        = dword ptr 8
.text:F82241C0
.text:F82241C0    sub     esp, 54h
.text:F82241C3    push    ebx
.text:F82241C4    xor     ebx, ebx
.text:F82241C6    push    esi
.text:F82241C7    mov     [esp+5Ch+ProcessObject], ebx
.text:F82241CB    call    KeGetCurrentIrql
.text:F82241D0    mov     esi, [esp+5Ch+ProcessHandle]
.text:F82241D4    cmp     al, 2          ;
.text:F82241D4                ; IRQL >= DISPATCH_LEVEL? Abort
.text:F82241D4                ; ( This is impossible for a system service )
.text:F82241D6    jnb     Ret_KavNtTerminateProcess
.text:F82241DC    cmp     esi, ebx          ;
.text:F82241DC                ; Null process handle? Abort
.text:F82241DE    jz      Ret_KavNtTerminateProcess
.text:F82241E4    call    PsGetCurrentProcessId
.text:F82241E9    mov     [esp+5Ch+ProcessData.CurrentProcessId], eax
.text:F82241ED    xor     eax, eax
.text:F82241EF    cmp     esi, 0FFFFFFFh
.text:F82241F2    push    esi          ; ProcessHandle
.text:F82241F3    setnz  al
.text:F82241F6    dec     eax
.text:F82241F7    mov     [esp+60h+ProcessData.TargetIsCurrentProcess], eax
.text:F82241FB    call    KavGetProcessIdFromProcessHandle
.text:F8224200    lea     ecx, [esp+5Ch+ProcessObject] ; Object
.text:F8224204    push    ebx          ; HandleInformation
.text:F8224205    push    ecx          ; Object
.text:F8224206    push    ebx          ; AccessMode
.text:F8224207    push    ebx          ; ObjectType
.text:F8224208    push    0F0000h      ; DesiredAccess
.text:F822420D    push    esi          ; Handle

```

```

.text:F822420E    mov     [esp+74h+ProcessData.TargetProcessId], eax
.text:F8224212    mov     [esp+74h+var_4], ebx
.text:F8224216    call    ds:ObReferenceObjectByHandle
.text:F822421C    test    eax, eax
.text:F822421E    jl      short loc_F8224246
.text:F8224220    mov     edx, [esp+5Ch+ProcessObject]
.text:F8224224    mov     eax, g_EprocessNameOffset
.text:F8224229    add     eax, edx
.text:F822422B    push    40h                ; size_t
.text:F822422D    lea     ecx, [esp+60h+ProcessData.ProcessName]
.text:F8224231    push    eax                ; char *
.text:F8224232    push    ecx                ; char *
.text:F8224233    call    ds:strncpy
.text:F8224239    mov     ecx, [esp+68h+ProcessObject]
.text:F822423D    add     esp, 0Ch
.text:F8224240    call    ds:ObfDereferenceObject
.text:F8224246
.text:F8224246    loc_F8224246:                ; CODE XREF: KavNtTerminateProcess+5Ej
.text:F8224246    cmp     esi, 0FFFFFFFh
.text:F8224249    jnz     short loc_F8224255
.text:F822424B    mov     edx, [esp+5Ch+ProcessData.TargetProcessId]
.text:F822424F    push    edx
.text:F8224250    call    sub_F8226710
.text:F8224255
.text:F8224255    loc_F8224255:                ; CODE XREF: KavNtTerminateProcess+89j
.text:F8224255    lea     eax, [esp+5Ch+ProcessData]
.text:F8224259    push    ebx                ; int
.text:F822425A    push    eax                ; ProcessData
.text:F822425B    call    KavCheckTerminateProcess
.text:F8224260    cmp     eax, 7
.text:F8224263    jz      short loc_F822427D
.text:F8224265    cmp     eax, 1
.text:F8224268    jz      short loc_F822427D
.text:F822426A    cmp     eax, ebx
.text:F822426C    jz      short loc_F822427D
.text:F822426E    mov     esi, STATUS_ACCESS_DENIED
.text:F8224273    mov     eax, esi
.text:F8224275    pop     esi
.text:F8224276    pop     ebx
.text:F8224277    add     esp, 54h
.text:F822427A    retn     8
.text:F822427D ; -----
.text:F822427D
.text:F822427D    loc_F822427D:                ; CODE XREF: KavNtTerminateProcess+A3j
.text:F822427D                ; KavNtTerminateProcess+A8j ...
.text:F822427D    mov     eax, [esp+5Ch+ProcessData.TargetProcessId]
.text:F8224281    cmp     eax, 1000h
.text:F8224286    jnb     short loc_F8224296
.text:F8224288    mov     dword_F8228460[eax*8], ebx
.text:F822428F    mov     byte_F8228464[eax*8], bl
.text:F8224296
.text:F8224296    loc_F8224296:                ; CODE XREF: KavNtTerminateProcess+C6j
.text:F8224296    push    eax
.text:F8224297    call    sub_F82134D0
.text:F822429C    mov     ecx, [esp+5Ch+ProcessData.TargetProcessId]
.text:F82242A0    push    ecx
.text:F82242A1    call    sub_F8221F70

```

```

.text:F82242A6    mov     edx, [esp+5Ch+ExitStatus]
.text:F82242AA    push    edx
.text:F82242AB    push    esi
.text:F82242AC    call    OrigNtTerminateProcess
.text:F82242B2    mov     esi, eax
.text:F82242B4    lea     eax, [esp+5Ch+ProcessData]
.text:F82242B8    push    1                ; int
.text:F82242BA    push    eax              ; ProcessData
.text:F82242BB    mov     [esp+64h+var_4], esi
.text:F82242BF    call    KavCheckTerminateProcess
.text:F82242C4    mov     eax, esi
.text:F82242C6    pop     esi
.text:F82242C7    pop     ebx
.text:F82242C8    add     esp, 54h
.text:F82242CB    retn    8
.text:F82242CE    ; -----
.text:F82242CE
.text:F82242CE Ret_KavNtTerminateProcess:                ; CODE XREF: KavNtTerminateProcess+16j
.text:F82242CE                ; KavNtTerminateProcess+1Ej
.text:F82242CE    mov     ecx, [esp+5Ch+ExitStatus]
.text:F82242D2    push    ecx
.text:F82242D3    push    esi
.text:F82242D4    call    OrigNtTerminateProcess
.text:F82242DA    pop     esi
.text:F82242DB    pop     ebx
.text:F82242DC    add     esp, 54h
.text:F82242DF    retn    8
.text:F82242DF KavNtTerminateProcess endp

```

The whole purpose of this particular system service hook is “shady” as well. The hook prevents certain KAV processes from being terminated, even by a legitimate computer administrator - something that is once again typically associated with malicious software, such as rootkits, rather than commercial software applications. One possible explanation for this is that it is an attempt to prevent viruses from terminating the virus scanner processes itself, although one wonders how much of a concern this would be if KAV’s real-time scanning mechanisms really do work as advertised.

Additionally, KAV appears to do some state tracking just before the process is terminated with this system service hook. The proper way to do this would have been through PsSetCreateProcessNotifyRoutine which is a documented kernel function that allows drivers to register a callback that is called on process creation and process exit.

3.1.5 Patching non-exported, non-system-service kernel functions

KAV’s kernel patching is not limited to just system services, however. One of the most dangerous hooks that KAV installs is one in the middle of the nt!SwapContext function, which is neither exported nor a system service (and

thus has no reliable mechanism to be detected by driver code, other than code fingerprinting). `nt!SwapContext` is called by the kernel on every context switch in order to perform some internal bookkeeping tasks.

Patching such a critical, non-exported kernel function with a mechanism as unreliable as blind code fingerprinting is, in the author's opinion, not a particularly good idea. To make matters worse, KAV actually modifies code in the middle of `nt!SwapContext` instead of patching the start of the function, and as such makes assumptions about the internal register and stack usage of this kernel function.

```
kd> u nt!SwapContext
nt!SwapContext:
804db924 0ac9          or      cl,cl
804db926 26c6462d02    mov     byte ptr es:[esi+0x2d],0x2
804db92b 9c            pushfd
804db92c 8b0b          mov     ecx,[ebx]
804db92e e9dd69d677    jmp     klif!KavSwapContext (f8242310)
```

The unmodified `nt!SwapContext` has code that runs along the lines of this:

```
lkd> u nt!SwapContext
nt!SwapContext:
80540ab0 0ac9          or      cl,cl
80540ab2 26c6462d02    mov     byte ptr es:[esi+0x2d],0x2
80540ab7 9c            pushfd
80540ab8 8b0b          mov     ecx,[ebx]
80540aba 83bb9409000000 cmp     dword ptr [ebx+0x994],0x0
80540ac1 51            push    ecx
80540ac2 0f8535010000 jne     nt!SwapContext+0x14d (80540bfd)
80540ac8 833d0ca0558000 cmp     dword ptr [nt!PPerfGlobalGroupMask (8055a00c)],0x0
```

This is an extremely dangerous patching operation to make, for several reasons:

1. `nt!SwapContext` is a *very* hot code path, as it is called on every single context switch. Therefore, patching it at runtime without running a non-trivial risk of bringing down the system is very difficult, especially on multiprocessor systems. KAV attempts to solve the synchronization problems relating to patching this function on uniprocessor systems by disabling interrupts entirely, but this approach will not work reliably on multiprocessor systems. KAV makes no attempt to address this problem on multiprocessor systems and puts them at the risk of randomly failing on boot during KAV's patching.
2. Reliably locating this function and making assumptions about the register and stack usage (and instruction layout) across all released and future Windows versions is a practical impossibility, and yet KAV attempts to do just this. This puts KAV customers at the mercy of the next Windows

update, which may cause their systems to crash on boot because KAV's hooking code makes an assumption that has been invalidated about the context-switching process.

Additionally, in order to perform code patching on the kernel, KAV adjusts the page protections of kernel code to be writable by altering PTE attributes directly instead of using documented functions (which would have proper locking semantics for accessing internal memory management structures).

KAV nt!SwapContext patching:

```
.text:F82264EA    mov     eax, 90909090h ; Build the code to be written to nt!SwapContext
.text:F82264EF    mov     [ebp+var_38], eax
.text:F82264F2    mov     [ebp+var_34], eax
.text:F82264F5    mov     [ebp+var_30], ax
.text:F82264F9    mov     byte ptr [ebp+var_38], 0E9h
.text:F82264FD    mov     ecx, offset KavSwapContext
.text:F8226502    sub     ecx, ebx
.text:F8226504    sub     ecx, 5
.text:F8226507    mov     [ebp+var_38+1], ecx
.text:F822650A    mov     ecx, [ebp+var_1C]
.text:F822650D    lea     edx, [ecx+ebx]
.text:F8226510    mov     dword_F8228338, edx
.text:F8226516    mov     esi, ebx
.text:F8226518    mov     edi, offset unk_F8227DBC
.text:F822651D    mov     eax, ecx
.text:F822651F    shr     ecx, 2
.text:F8226522    rep movsd
.text:F8226524    mov     ecx, eax
.text:F8226526    and     ecx, 3
.text:F8226529    rep movsb
.text:F822652B    lea     ecx, [ebp+var_48] ; Make nt!SwapContext writable by directly accessing
                        ; the PTEs.
.text:F822652E    push    ecx
.text:F822652F    push    1
.text:F8226531    push    ebx
.text:F8226532    call    ModifyPteAttributes
.text:F8226537    test    al, al
.text:F8226539    jz      short loc_F8226588
.text:F822653B    mov     ecx, offset KavInternalSpinLock
.text:F8226540    call    KavSpinLockAcquire ; Disable interrupts
.text:F8226545    mov     ecx, [ebp+var_1C] ; Write to kernel code
.text:F8226548    lea     esi, [ebp+var_38]
.text:F822654B    mov     edi, ebx
.text:F822654D    mov     edx, ecx
.text:F822654F    shr     ecx, 2
.text:F8226552    rep movsd
.text:F8226554    mov     ecx, edx
.text:F8226556    and     ecx, 3
.text:F8226559    rep movsb
.text:F822655B    mov     edx, eax
.text:F822655D    mov     ecx, offset KavInternalSpinLock
.text:F8226562    call    KavSpinLockRelease ; Reenable interrupts
.text:F8226567    lea     eax, [ebp+var_48] ; Restore the original PTE attributes.
```

```

.text:F822656A    push    eax
.text:F822656B    mov     ecx, [ebp+var_48]
.text:F822656E    push    ecx
.text:F822656F    push    ebx
.text:F8226570    call   ModifyPteAttributes
.text:F8226575    mov     al, 1
.text:F8226577    mov     ecx, [ebp+var_10]
.text:F822657A    mov     large fs:0, ecx
.text:F8226581    pop     edi
.text:F8226582    pop     esi
.text:F8226583    pop     ebx
.text:F8226584    mov     esp, ebp
.text:F8226586    pop     ebp
.text:F8226587    retn

```

KavSpinLockAcquire subroutine (disables interrupts):

```

.text:F8221240 KavSpinLockAcquire proc near          ; CODE XREF: sub_F8225690+D7p
.text:F8221240          ; sub_F8225D50+8Cp ...
.text:F8221240    pushf
.text:F8221241    pop     eax
.text:F8221242
.text:F8221242 loc_F8221242:                          ; CODE XREF: KavSpinLockAcquire+13j
.text:F8221242    cli
.text:F8221243    lock bts dword ptr [ecx], 0
.text:F8221248    jb     short loc_F822124B
.text:F822124A    retn
.text:F822124B ; -----
.text:F822124B
.text:F822124B loc_F822124B:                          ; CODE XREF: KavSpinLockAcquire+8j
.text:F822124B    push    eax
.text:F822124C    popf
.text:F822124D
.text:F822124D loc_F822124D:                          ; CODE XREF: KavSpinLockAcquire+17j
.text:F822124D    test    dword ptr [ecx], 1
.text:F8221253    jz     short loc_F8221242
.text:F8221255    pause
.text:F8221257    jmp     short loc_F822124D
.text:F8221257 KavSpinLockAcquire endp

```

KavSpinLockRelease subroutine (reenables interrupts):

```

.text:F8221260 KavSpinLockRelease proc near          ; CODE XREF: sub_F8225690+F2p
.text:F8221260          ; sub_F8225D50+BAp ...
.text:F8221260    mov     dword ptr [ecx], 0
.text:F8221266    push    edx
.text:F8221267    popf
.text:F8221268    retn
.text:F8221268 KavSpinLockRelease endp

```

ModifyPteAttributes subroutine:

```

.text:F82203C0 ModifyPteAttributes proc near          ; CODE XREF: sub_F821A9D0+91p

```

```

.text:F82203C0                                     ; sub_F8220950+43p ...
.text:F82203C0
.text:F82203C0 var_24          = dword ptr -24h
.text:F82203C0 var_20          = byte ptr -20h
.text:F82203C0 var_1C          = dword ptr -1Ch
.text:F82203C0 var_18          = dword ptr -18h
.text:F82203C0 var_10          = dword ptr -10h
.text:F82203C0 var_4           = dword ptr -4
.text:F82203C0 arg_0           = dword ptr  8
.text:F82203C0 arg_4           = byte ptr  0Ch
.text:F82203C0 arg_8           = dword ptr  10h
.text:F82203C0
.text:F82203C0    push    ebp
.text:F82203C1    mov     ebp, esp
.text:F82203C3    push    0FFFFFFFh
.text:F82203C5    push    offset dword_F8212180
.text:F82203CA    push    offset _except_handler3
.text:F82203CF    mov     eax, large fs:0
.text:F82203D5    push    eax
.text:F82203D6    mov     large fs:0, esp
.text:F82203DD    sub     esp, 14h
.text:F82203E0    push    ebx
.text:F82203E1    push    esi
.text:F82203E2    push    edi
.text:F82203E3    mov     [ebp+var_18], esp
.text:F82203E6    xor     ebx, ebx
.text:F82203E8    mov     [ebp+var_20], bl
.text:F82203EB    mov     esi, [ebp+arg_0]
.text:F82203EE    mov     ecx, esi
.text:F82203F0    call    KavGetEFlags
.text:F82203F5    push    esi
.text:F82203F6    call    KavGetPte          ; This is a function pointer filled in at runtime,
                                ; differing based on whether the system has PAE
                                ; enabled or not.
.text:F82203FC    mov     edi, eax
.text:F82203FE    mov     [ebp+var_1C], edi
.text:F8220401    cmp     edi, 0FFFFFFFh
.text:F8220404    jz      short loc_F8220458
.text:F8220406    mov     [ebp+var_4], ebx
.text:F8220409    mov     ecx, esi
.text:F822040B    call    KavGetEFlags
.text:F8220410    mov     eax, [edi]
.text:F8220412    test    al, 1
.text:F8220414    jz      short loc_F8220451
.text:F8220416    mov     ecx, eax
.text:F8220418    mov     [ebp+var_24], ecx
.text:F822041B    cmp     [ebp+arg_4], bl
.text:F822041E    jz      short loc_F8220429
.text:F8220420    mov     eax, [ebp+var_1C]
.text:F8220423    lock or dword ptr [eax], 2
.text:F8220427    jmp     short loc_F8220430
.text:F8220429 ; -----
.text:F8220429
.text:F8220429 loc_F8220429:          ; CODE XREF: ModifyPteAttributes+5Ej
.text:F8220429    mov     eax, [ebp+var_1C]
.text:F822042C    lock and dword ptr [eax], 0FFFFFFDh
.text:F8220430

```

```

.text:F8220430 loc_F8220430:                ; CODE XREF: ModifyPteAttributes+67j
.text:F8220430      mov     eax, [ebp+arg_8]
.text:F8220433      cmp     eax, ebx
.text:F8220435      jz      short loc_F822043C
.text:F8220437      and     ecx, 2
.text:F822043A      mov     [eax], cl
.text:F822043C
.text:F822043C loc_F822043C:                ; CODE XREF: ModifyPteAttributes+75j
.text:F822043C      mov     [ebp+var_20], 1
.text:F8220440      mov     eax, [ebp+arg_0]
.text:F8220443      invlpg  byte ptr [eax]
.text:F8220446      jmp     short loc_F8220451
.text:F8220448 ; -----
.text:F8220448
.text:F8220448 loc_F8220448:                ; DATA XREF: .text:F8212184o
.text:F8220448      mov     eax, 1
.text:F822044D      retn
.text:F822044E ; -----
.text:F822044E
.text:F822044E loc_F822044E:                ; DATA XREF: .text:F8212188o
.text:F822044E      mov     esp, [ebp-18h]
.text:F8220451
.text:F8220451 loc_F8220451:                ; CODE XREF: ModifyPteAttributes+54j
.text:F8220451      ; ModifyPteAttributes+86j
.text:F8220451      mov     [ebp+var_4], 0FFFFFFFh
.text:F8220458
.text:F8220458 loc_F8220458:                ; CODE XREF: ModifyPteAttributes+44j
.text:F8220458      mov     al, [ebp+var_20]
.text:F822045B      mov     ecx, [ebp+var_10]
.text:F822045E      mov     large fs:0, ecx
.text:F8220465      pop     edi
.text:F8220466      pop     esi
.text:F8220467      pop     ebx
.text:F8220468      mov     esp, ebp
.text:F822046A      pop     ebp
.text:F822046B      retn     0Ch
.text:F822046B ModifyPteAttributes endp

```

3.1.6 Allowing User-mode Code to Access Kernel Memory

One of the most important principles of the kernel/user division that modern operating systems enforce is that user mode is not allowed to directly access kernel mode memory. This is necessary to enforce system stability, such as to prevent a buggy user mode program from corrupting the kernel and bringing down the whole system. Unfortunately, the KAV programmers appear to think that this distinction is not really so important after all.

One of the strangest of the unsafe practices implemented by KAV is to allow user mode to directly call some portions of their kernel driver (within kernel address space!) instead of just loading a user mode DLL (or otherwise loading user mode code in the target process).

This mechanism appears to be used to inspect DLLs as they are loaded - a task which would be much better accomplished with PsSetLoadImageNotifyRoutine.

KAV patches kernel32.dll as a new process is created, such that the export table points all of the DLL-loading routines (e.g. LoadLibraryA) to a thunk that calls portions of KAV's driver in kernel mode. Additionally, KAV modifies protections on parts of its code and data sections to allow user mode read access.

KAV sets a PsLoadImageNotifyRoutine hook to detect kernel32.dll being loaded in order to know when to patch kernel32's export table. The author wonders why KAV did not just do their work from within PsSetLoadImageNotifyRoutine directly instead of going through all the trouble to allow user mode to call kernel mode for a LoadLibrary hook.

The CheckInjectCodeForNewProcess function is called when a new process loads an image, and checks for kernel32 being loaded. If this is the case, it will queue an APC to the process that will perform patching.

```
.text:F82218B0 ; int __stdcall CheckInjectCodeForNewProcess(wchar_t *,PUCHAR ImageBase)
.text:F82218B0 CheckInjectCodeForNewProcess proc near ; CODE XREF: KavLoadImageNotifyRoutine+B5p
.text:F82218B0 ; KavDoKernel32Check+41p
.text:F82218B0
.text:F82218B0 arg_0 = dword ptr 4
.text:F82218B0 ImageBase = dword ptr 8
.text:F82218B0
.text:F82218B0 mov al, byte_F82282F9
.text:F82218B5 push esi
.text:F82218B6 test al, al
.text:F82218B8 push edi
.text:F82218B9 jz short loc_F8221936
.text:F82218BB mov eax, [esp+8+arg_0]
.text:F82218BF push offset aKernel32_dll ; "kernel32.dll"
.text:F82218C4 push eax ; wchar_t *
.text:F82218C5 call ds:_wcsicmp
.text:F82218CB add esp, 8
.text:F82218CE test eax, eax
.text:F82218D0 jnz short loc_F8221936
.text:F82218D2 mov al, g_FoundKernel32Exports
.text:F82218D7 mov edi, [esp+8+ImageBase]
.text:F82218DB test al, al
.text:F82218DD jnz short KavInitializePatchApcLabel
.text:F82218DF push edi
.text:F82218E0 call KavCheckFindKernel32Exports
.text:F82218E5 test al, al
.text:F82218E7 jz short loc_F8221936
.text:F82218E9
.text:F82218E9 KavInitializePatchApcLabel: ; CODE XREF: CheckInjectCodeForNewProcess+2Dj
.text:F82218E9 push '3SeB' ; Tag
.text:F82218EE push 30h ; NumberOfBytes
.text:F82218F0 push 0 ; PoolType
.text:F82218F2 call ds:ExAllocatePoolWithTag
.text:F82218F8 mov esi, eax
.text:F82218FA test esi, esi
.text:F82218FC jz short loc_F8221936
```

```

.text:F82218FE    push    edi
.text:F82218FF    push    0
.text:F8221901    push    offset KavPatchNewProcessApcRoutine
.text:F8221906    push    offset loc_F82218A0
.text:F822190B    push    offset loc_F8221890
.text:F8221910    push    0
.text:F8221912    call    KeGetCurrentThread
.text:F8221917    push    eax
.text:F8221918    push    esi
.text:F8221919    call    KeInitializeApc
.text:F822191E    push    0
.text:F8221920    push    0
.text:F8221922    push    0
.text:F8221924    push    esi
.text:F8221925    call    KeInsertQueueApc
.text:F822192B    test    al, al
.text:F822192D    jnz     short loc_F822193D
.text:F822192F    push    esi                ; P
.text:F8221930    call    ds:ExFreePool
.text:F8221936
.text:F8221936    loc_F8221936:                ; CODE XREF: CheckInjectCodeForNewProcess+9j
.text:F8221936                ; CheckInjectCodeForNewProcess+20j ...
.text:F8221936    pop     edi
.text:F8221937    xor     al, al
.text:F8221939    pop     esi
.text:F822193A    retn    8
.text:F822193D    ; -----
.text:F822193D
.text:F822193D    loc_F822193D:                ; CODE XREF: CheckInjectCodeForNewProcess+7Dj
.text:F822193D    pop     edi
.text:F822193E    mov     al, 1
.text:F8221940    pop     esi
.text:F8221941    retn    8

```

The APC routine itself patches kernel32's export table (and generates the thunks to call kernel mode) and adjusts PTE attributes on KAV's driver image to allow user mode access.

```

.text:F8221810    KavPatchNewProcessApcRoutine proc near ; DATA XREF: CheckInjectCodeForNewProcess+510
.text:F8221810
.text:F8221810    var_8            = dword ptr -8
.text:F8221810    var_4            = dword ptr -4
.text:F8221810    ImageBase        = dword ptr 8
.text:F8221810
.text:F8221810    push    ebp
.text:F8221811    mov     ebp, esp
.text:F8221813    sub     esp, 8
.text:F8221816    mov     eax, [ebp+ImageBase]
.text:F8221819    push    esi
.text:F822181A    push    eax                ; ImageBase
.text:F822181B    call    KavPatchImageForNewProcess
.text:F8221820    mov     esi, dword_F8230518
.text:F8221826    mov     eax, dword_F823051C
.text:F822182B    and     esi, 0FFFFFF00h
.text:F8221831    cmp     esi, eax

```

```

.text:F8221833    mov     [ebp+ImageBase], esi
.text:F8221836    jnb     short loc_F8221883
.text:F8221838
.text:F8221838    loc_F8221838:                ; CODE XREF: KavPatchNewProcessApcRoutine+71j
.text:F8221838    push    esi
.text:F8221839    call    KavPageTranslation0
.text:F822183F    push    esi
.text:F8221840    mov     [ebp+var_8], eax
.text:F8221843    call    KavPageTranslation1
.text:F8221849    mov     [ebp+var_4], eax
.text:F822184C    mov     eax, [ebp+var_8]
.text:F822184F    lock or dword ptr [eax], 4
.text:F8221853    lock and dword ptr [eax], 0FFFFFFFh
.text:F822185A    mov     eax, [ebp+var_4]
.text:F822185D    invlpg  byte ptr [eax]
.text:F8221860    lock or dword ptr [eax], 4
.text:F8221864    lock and dword ptr [eax], 0FFFFFFDh
.text:F822186B    mov     eax, [ebp+ImageBase]
.text:F822186E    invlpg  byte ptr [eax]
.text:F8221871    mov     eax, dword_F823051C
.text:F8221876    add     esi, 1000h
.text:F822187C    cmp     esi, eax
.text:F822187E    mov     [ebp+ImageBase], esi
.text:F8221881    jnb     short loc_F8221838
.text:F8221883
.text:F8221883    loc_F8221883:                ; CODE XREF: KavPatchNewProcessApcRoutine+26j
.text:F8221883    pop     esi
.text:F8221884    mov     esp, ebp
.text:F8221886    pop     ebp
.text:F8221887    retn    0Ch
.text:F8221887    KavPatchNewProcessApcRoutine endp

.text:F8221750    ; int __stdcall KavPatchImageForNewProcess(PUCHAR ImageBase)
.text:F8221750    KavPatchImageForNewProcess proc near    ; CODE XREF: KavPatchNewProcessApcRoutine+Bp
.text:F8221750
.text:F8221750    ImageBase      = dword ptr 8
.text:F8221750
.text:F8221750    push    ebx
.text:F8221751    call    ds:KeEnterCriticalRegion
.text:F8221757    mov     eax, dword_F82282F4
.text:F822175C    push    1                ; Wait
.text:F822175E    push    eax                ; Resource
.text:F822175F    call    ds:ExAcquireResourceExclusiveLite
.text:F8221765    push    1
.text:F8221767    call    KavSetPageAttributes1
.text:F822176C    mov     ecx, [esp+ImageBase]
.text:F8221770    push    ecx                ; ImageBase
.text:F8221771    call    KavPatchImage
.text:F8221776    push    0
.text:F8221778    mov     bl, al
.text:F822177A    call    KavSetPageAttributes1
.text:F822177F    mov     ecx, dword_F82282F4 ; Resource
.text:F8221785    call    ds:ExReleaseResourceLite
.text:F822178B    call    ds:KeLeaveCriticalRegion
.text:F8221791    mov     al, bl
.text:F8221793    pop     ebx
.text:F8221794    retn    4

```

```
.text:F8221794 KavPatchImageForNewProcess endp
```

The actual image patching reprotects the export table of kernel32, changes the export address table entries for the LoadLibrary* family of functions to point to a thunk that is written into spare space within the kernel32 image, and writes the actual thunk code out:

```
.text:F8221680 ; int __stdcall KavPatchImage(PUCHAR ImageBase)
.text:F8221680 KavPatchImage    proc near                ; CODE XREF: KavPatchImageForNewProcess+21p
.text:F8221680
.text:F8221680 var_C            = dword ptr -0Ch
.text:F8221680 FunctionVa      = dword ptr -8
.text:F8221680 var_4          = dword ptr -4
.text:F8221680 ImageBase       = dword ptr  4
.text:F8221680
.text:F8221680    mov     eax, [esp+ImageBase]
.text:F8221684    sub     esp, 0Ch
.text:F8221687    push    ebp
.text:F8221688    push    3Ch
.text:F822168A    push    eax
.text:F822168B    call    KavReprotectExportTable
.text:F8221690    mov     ebp, eax
.text:F8221692    test    ebp, ebp
.text:F8221694    jnz     short loc_F822169F
.text:F8221696    xor     al, al
.text:F8221698    pop     ebp
.text:F8221699    add     esp, 0Ch
.text:F822169C    retn     4
.text:F822169F ; -----
.text:F822169F
.text:F822169F loc_F822169F:                                ; CODE XREF: KavPatchImage+14j
.text:F822169F    push    ebx
.text:F82216A0    push    esi
.text:F82216A1    push    edi
.text:F82216A2    xor     ebx, ebx
.text:F82216A4    mov     edi, ebp
.text:F82216A6    mov     esi, offset ExportedFunctionsToCheckTable
.text:F82216AB
.text:F82216AB CheckNextFunctionInTable:                        ; CODE XREF: KavPatchImage+B4j
.text:F82216AB    mov     edx, [esi+0Ch]
.text:F82216AE    mov     eax, [esp+1Ch+ImageBase]
.text:F82216B2    lea     ecx, [esp+1Ch+var_C]
.text:F82216B6    push    ecx
.text:F82216B7    push    edx
.text:F82216B8    push    eax
.text:F82216B9    call    LookupExportedFunction
.text:F82216BE    test    eax, eax
.text:F82216C0    mov     [esp+1Ch+FunctionVa], eax
.text:F82216C4    jz      short loc_F8221725
.text:F82216C6    mov     edx, [esp+1Ch+var_C]
.text:F82216CA    lea     ecx, [esp+1Ch+var_4]
.text:F82216CE    push    ecx
.text:F82216CF    push    40h
.text:F82216D1    push    4
.text:F82216D3    push    edx
```

```

.text:F82216D4    call    KavExecuteNtProtectVirtualMemoryInt2E
.text:F82216D9    test    al, al
.text:F82216DB    jz      short loc_F8221725
.text:F82216DD    cmp     dword ptr [esi], 0
.text:F82216E0    jnz     short loc_F82216EF
.text:F82216E2    mov     eax, [esp+1Ch+FunctionVa]
.text:F82216E6    mov     ecx, [esp+1Ch+var_C]
.text:F82216EA    mov     [esi], eax
.text:F82216EC    mov     [esi+8], ecx
.text:F82216EF
.text:F82216EF    loc_F82216EF:                ; CODE XREF: KavPatchImage+60j
.text:F82216EF    mov     eax, edi
.text:F82216F1    mov     edx, 90909090h
.text:F82216F6    mov     [eax], edx
.text:F82216F8    mov     [eax+4], edx
.text:F82216FB    mov     [eax+8], edx
.text:F82216FE    mov     [eax+0Ch], dx
.text:F8221702    mov     [eax+0Eh], dl
.text:F8221705    mov     byte ptr [edi], 0E9h
.text:F8221708    mov     ecx, [esi+4]
.text:F822170B    mov     edx, ebx
.text:F822170D    sub     ecx, ebx
.text:F822170F    sub     ecx, ebp
.text:F8221711    sub     ecx, 5
.text:F8221714    mov     [edi+1], ecx
.text:F8221717    mov     ecx, [esp+1Ch+ImageBase]
.text:F822171B    mov     eax, [esp+1Ch+var_C]
.text:F822171F    sub     edx, ecx
.text:F8221721    add     edx, ebp
.text:F8221723    mov     [eax], edx          ;
.text:F8221723    ; Patching Export Table here
.text:F8221723    ; e.g. write to 7c802f58
.text:F8221723    ; (kernel32 EAT entry for LoadLibraryA)
.text:F8221723    ;
.text:F8221723    ;          578  241 00001D77 LoadLibraryA = _LoadLibraryA@4
.text:F8221723    ;          579  242 00001D4F LoadLibraryExA = _LoadLibraryExA@12
.text:F8221723    ;          580  243 00001AF1 LoadLibraryExW = _LoadLibraryExW@12
.text:F8221723    ;          581  244 0000ACD3 LoadLibraryW = _LoadLibraryW@4
.text:F8221723    ;
.text:F8221723    ; KAV writes a new RVA pointing to its hook code here.
.text:F8221725
.text:F8221725    loc_F8221725:                ; CODE XREF: KavPatchImage+44j
.text:F8221725    ; KavPatchImage+5Bj
.text:F8221725    add     esi, 10h
.text:F8221728    add     ebx, 0Fh
.text:F822172B    add     edi, 0Fh
.text:F822172E    cmp     esi, offset byte_F82357E0
.text:F8221734    jb      CheckNextFunctionInTable
.text:F822173A    pop     edi
.text:F822173B    pop     esi
.text:F822173C    pop     ebx
.text:F822173D    mov     al, 1
.text:F822173F    pop     ebp
.text:F8221740    add     esp, 0Ch
.text:F8221743    retn    4
.text:F8221743    KavPatchImage    endp

```

KAV's export table reprotecting code assumes that the user mode PE header is well-formed and does not contain offsets pointing to kernel mode addresses:

```
.text:F8221360 KavReprotectExportTable proc near          ; CODE XREF: KavPatchImage+Bp
.text:F8221360
.text:F8221360 var_10          = dword ptr -10h
.text:F8221360 var_C          = dword ptr -0Ch
.text:F8221360 var_8          = dword ptr -8
.text:F8221360 var_4          = dword ptr -4
.text:F8221360 arg_0          = dword ptr  4
.text:F8221360 arg_4          = dword ptr  8
.text:F8221360
.text:F8221360 mov     eax, [esp+arg_0]
.text:F8221364 sub     esp, 10h
.text:F8221367 cmp     word ptr [eax], 'ZM'
.text:F822136C push    ebx
.text:F822136D push    ebp
.text:F822136E push    esi
.text:F822136F push    edi
.text:F8221370 jnz     loc_F8221442
.text:F8221376 mov     esi, [eax+3Ch]
.text:F8221379 add     esi, eax
.text:F822137B mov     [esp+20h+var_C], esi
.text:F822137F cmp     dword ptr [esi], 'EP'
.text:F8221385 jnz     loc_F8221442
.text:F822138B lea     eax, [esp+20h+var_8]
.text:F822138F xor     edx, edx
.text:F8221391 mov     dx, [esi+14h]
.text:F8221395 push    eax
.text:F8221396 xor     eax, eax
.text:F8221398 push    40h
.text:F822139A mov     ax, [esi+6]
.text:F822139E lea     ecx, [eax+eax*4]
.text:F82213A1 lea     eax, [edx+ecx*8+18h]
.text:F82213A5 push    eax
.text:F82213A6 push    esi
.text:F82213A7 call    KavExecuteNtProtectVirtualMemoryInt2E ; NtProtectVirtualMemory
.text:F82213AC test    al, al
.text:F82213AE jz      loc_F8221442
.text:F82213B4 mov     ecx, [esi+8]
.text:F82213B7 mov     [esp+20h+var_10], 0
.text:F82213BF inc     ecx
.text:F82213C0 mov     [esi+8], ecx
.text:F82213C3 xor     ecx, ecx
.text:F82213C5 mov     cx, [esi+14h]
.text:F82213C9 cmp     word ptr [esi+6], 0
.text:F82213CE lea     edi, [ecx+esi+18h]
.text:F82213D2 jbe     short loc_F8221442
.text:F82213D4 mov     ebp, [esp+20h+arg_4]
.text:F82213D8
.text:F82213D8 loc_F82213D8:          ; CODE XREF: KavReprotectExportTable+E0j
.text:F82213D8 mov     ebx, [edi+10h]
.text:F82213DB test    ebx, 0FFFh
.text:F82213E1 jz      short loc_F82213EA
.text:F82213E3 or      ebx, 0FFFh
.text:F82213E9 inc     ebx
.text:F82213EA
```

```

.text:F82213EA loc_F82213EA:                ; CODE XREF: KavReprotectExportTable+81j
.text:F82213EA    mov     ecx, [edi+8]
.text:F82213ED    mov     edx, ebx
.text:F82213EF    sub     edx, ecx
.text:F82213F1    cmp     edx, ebp
.text:F82213F3    jle     short loc_F822142C
.text:F82213F5    mov     esi, [edi+0Ch]
.text:F82213F8    mov     ecx, [esp+20h+arg_0]
.text:F82213FC    sub     esi, ebp
.text:F82213FE    push    ebp
.text:F82213FF    add     esi, ebx
.text:F8221401    add     esi, ecx
.text:F8221403    push    esi
.text:F8221404    call    KavFindSectionName
.text:F8221409    test    al, al
.text:F822140B    jz      short loc_F8221428
.text:F822140D    cmp     dword ptr [edi+1], 'TINI'
.text:F8221414    jz      short loc_F8221428
.text:F8221416    lea     eax, [esp+20h+var_4]
.text:F822141A    push    eax
.text:F822141B    push    40h
.text:F822141D    push    ebp
.text:F822141E    push    esi
.text:F822141F    call    KavExecuteNtProtectVirtualMemoryInt2E ; NtProtectVirtualMemory
.text:F8221424    test    al, al
.text:F8221426    jnz     short loc_F822144E
.text:F8221428
.text:F8221428 loc_F8221428:                ; CODE XREF: KavReprotectExportTable+ABj
.text:F8221428    ; KavReprotectExportTable+B4j
.text:F8221428    mov     esi, [esp+20h+var_C]
.text:F822142C
.text:F822142C loc_F822142C:                ; CODE XREF: KavReprotectExportTable+93j
.text:F822142C    mov     eax, [esp+20h+var_10]
.text:F8221430    xor     ecx, ecx
.text:F8221432    mov     cx, [esi+6]
.text:F8221436    add     edi, 28h
.text:F8221439    inc     eax
.text:F822143A    cmp     eax, ecx
.text:F822143C    mov     [esp+20h+var_10], eax
.text:F8221440    jb      short loc_F82213D8
.text:F8221442
.text:F8221442 loc_F8221442:                ; CODE XREF: KavReprotectExportTable+10j
.text:F8221442    ; KavReprotectExportTable+25j ...
.text:F8221442    pop     edi
.text:F8221443    pop     esi
.text:F8221444    pop     ebp
.text:F8221445    xor     eax, eax
.text:F8221447    pop     ebx
.text:F8221448    add     esp, 10h
.text:F822144B    retn    8
.text:F822144E ; -----
.text:F822144E
.text:F822144E loc_F822144E:                ; CODE XREF: KavReprotectExportTable+C6j
.text:F822144E    mov     eax, [edi+8]
.text:F8221451    mov     [edi+10h], ebx
.text:F8221454    add     eax, ebp
.text:F8221456    mov     [edi+8], eax

```

```

.text:F8221459    mov     eax, esi
.text:F822145B    pop     edi
.text:F822145C    pop     esi
.text:F822145D    pop     ebp
.text:F822145E    pop     ebx
.text:F822145F    add     esp, 10h
.text:F8221462    retn    8
.text:F8221462 KavReprotectExportTable endp

```

The mechanism that KAV uses to reprotect user mode code is much of a hack as well. KAV dynamically determines the system call ordinal of the NtProtectVirtualMemory system service and uses its own int 2e thunk to call the service.

```

.text:F8221320 KavExecuteNtProtectVirtualMemoryInt2E proc near
.text:F8221320                ; CODE XREF: KavReprotectExportTable+47p
.text:F8221320                ; KavReprotectExportTable+BFp ...
.text:F8221320
.text:F8221320 arg_0          = dword ptr 4
.text:F8221320 arg_4          = dword ptr 8
.text:F8221320 arg_8          = dword ptr 0Ch
.text:F8221320 arg_C          = dword ptr 10h
.text:F8221320
.text:F8221320    mov     eax, [esp+arg_0]
.text:F8221324    mov     ecx, [esp+arg_C]
.text:F8221328    mov     edx, [esp+arg_8]
.text:F822132C    push    ebx
.text:F822132D    mov     [esp+4+arg_0], eax
.text:F8221331    push    ecx
.text:F8221332    lea     eax, [esp+8+arg_4]
.text:F8221336    push    edx
.text:F8221337    mov     edx, NtProtectVirtualMemoryOrdinal
.text:F822133D    lea     ecx, [esp+0Ch+arg_0]
.text:F8221341    push    eax
.text:F8221342    push    ecx
.text:F8221343    push    0FFFFFFFh
.text:F8221345    push    edx
.text:F8221346    xor     bl, bl
.text:F8221348    call    KavInt2E
.text:F822134D    test    eax, eax
.text:F822134F    mov     al, 1
.text:F8221351    jge     short loc_F8221355
.text:F8221353    mov     al, bl
.text:F8221355
.text:F8221355 loc_F8221355:                ; CODE XREF: KavExecuteNtProtectVirtualMemoryInt2E+31j
.text:F8221355    pop     ebx
.text:F8221356    retn    10h
.text:F8221356 KavExecuteNtProtectVirtualMemoryInt2E endp

.user:F8231090 KavInt2E                proc near                ; CODE XREF: KavExecuteNtProtectVirtualMemoryInt2E+28p
.user:F8231090
.user:F8231090 arg_0          = dword ptr 8
.user:F8231090 arg_4          = dword ptr 0Ch
.user:F8231090

```

```

.user:F8231090    push    ebp
.user:F8231091    mov     ebp, esp
.user:F8231093    mov     eax, [ebp+arg_0]
.user:F8231096    lea     edx, [ebp+arg_4]
.user:F823109C    int     2Eh
.user:F823109C
.user:F823109E    pop     ebp
.user:F823109F    retn    18h
.user:F823109F   KavInt2E    endp
.user:F823109F

```

KAV's export lookup code does not correctly validate offsets garnered from the PE header before using them:

```

.text:F8220CA0 LookupExportedFunction proc near          ; CODE XREF: sub_F8217A60+C9p
.text:F8220CA0                                     ; sub_F82181D0+Dp ...
.text:F8220CA0
.text:F8220CA0 var_20          = dword ptr -20h
.text:F8220CA0 var_1C          = dword ptr -1Ch
.text:F8220CA0 var_18          = dword ptr -18h
.text:F8220CA0 var_14          = dword ptr -14h
.text:F8220CA0 var_10          = dword ptr -10h
.text:F8220CA0 var_C           = dword ptr -0Ch
.text:F8220CA0 var_8           = dword ptr -8
.text:F8220CA0 var_4           = dword ptr -4
.text:F8220CA0 arg_0           = dword ptr  4
.text:F8220CA0 arg_4           = dword ptr  8
.text:F8220CA0 arg_8           = dword ptr 0Ch
.text:F8220CA0
.text:F8220CA0    mov     edx, [esp+arg_0]
.text:F8220CA4    sub     esp, 20h
.text:F8220CA7    cmp     word ptr [edx], 'ZM'
.text:F8220CAC    push    ebx
.text:F8220CAD    push    ebp
.text:F8220CAE    push    esi
.text:F8220CAF    push    edi
.text:F8220CB0    jnz     loc_F8220DE1
.text:F8220CB6    mov     eax, [edx+3Ch]
.text:F8220CB9    add     eax, edx
.text:F8220CBB    cmp     dword ptr [eax], 'EP'
.text:F8220CC1    jnz     loc_F8220DE1
.text:F8220CC7    mov     eax, [eax+78h]
.text:F8220CCA    mov     edi, [esp+30h+arg_4]
.text:F8220CCE    add     eax, edx
.text:F8220CD0    mov     [esp+30h+var_14], eax
.text:F8220CD4    mov     esi, [eax+1Ch]
.text:F8220CD7    mov     ebx, [eax+24h]
.text:F8220CDA    mov     ecx, [eax+20h]
.text:F8220CDD    add     esi, edx
.text:F8220CDF    add     ebx, edx
.text:F8220CE1    add     ecx, edx
.text:F8220CE3    cmp     edi, 1000h
.text:F8220CE9    mov     [esp+30h+var_4], esi
.text:F8220CED    mov     [esp+30h+var_C], ebx
.text:F8220CF1    mov     [esp+30h+var_18], ecx
.text:F8220CF5    jnb     short loc_F8220D27

```

```

.text:F8220CF7    mov     ecx, [eax+10h]
.text:F8220CFA    mov     eax, edi
.text:F8220CFC    sub     eax, ecx
.text:F8220CFE    mov     eax, [esi+eax*4]
.text:F8220D01    add     eax, edx
.text:F8220D03    mov     edx, [esp+30h+arg_8]
.text:F8220D07    test    edx, edx
.text:F8220D09    jz      loc_F8220DE3
.text:F8220D0F    mov     ebx, ecx
.text:F8220D11    shl     ebx, 1Eh
.text:F8220D14    sub     ebx, ecx
.text:F8220D16    add     ebx, edi
.text:F8220D18    pop     edi
.text:F8220D19    lea     ecx, [esi+ebx*4]
.text:F8220D1C    pop     esi
.text:F8220D1D    pop     ebp
.text:F8220D1E    mov     [edx], ecx
.text:F8220D20    pop     ebx
.text:F8220D21    add     esp, 20h
.text:F8220D24    retn    0Ch
.text:F8220D27    ; -----
.text:F8220D27
.text:F8220D27    loc_F8220D27:                ; CODE XREF: LookupExportedFunction+55j
.text:F8220D27    mov     edi, [eax+14h]
.text:F8220D2A    mov     [esp+30h+arg_0], 0
.text:F8220D32    test    edi, edi
.text:F8220D34    mov     [esp+30h+var_8], edi
.text:F8220D38    jbe     loc_F8220DE1
.text:F8220D3E    mov     [esp+30h+var_1C], esi
.text:F8220D42
.text:F8220D42    loc_F8220D42:                ; CODE XREF: LookupExportedFunction+13Bj
.text:F8220D42    cmp     dword ptr [esi], 0
.text:F8220D45    jz      short loc_F8220DC5
.text:F8220D47    mov     ecx, [eax+18h]
.text:F8220D4A    xor     ebp, ebp
.text:F8220D4C    test    ecx, ecx
.text:F8220D4E    mov     [esp+30h+var_10], ecx
.text:F8220D52    jbe     short loc_F8220DC5
.text:F8220D54    mov     edi, [esp+30h+var_18]
.text:F8220D58    mov     [esp+30h+var_20], ebx
.text:F8220D5C
.text:F8220D5C    loc_F8220D5C:                ; CODE XREF: LookupExportedFunction+11Bj
.text:F8220D5C    mov     ebx, [esp+30h+var_20]
.text:F8220D60    xor     esi, esi
.text:F8220D62    mov     si, [ebx]
.text:F8220D65    mov     ebx, [esp+30h+arg_0]
.text:F8220D69    cmp     esi, ebx
.text:F8220D6B    jnz     short loc_F8220DAA
.text:F8220D6D    mov     eax, [edi]
.text:F8220D6F    mov     esi, [esp+30h+arg_4]
.text:F8220D73    add     eax, edx
.text:F8220D75
.text:F8220D75    loc_F8220D75:                ; CODE XREF: LookupExportedFunction+F3j
.text:F8220D75    mov     bl, [eax]
.text:F8220D77    mov     cl, bl
.text:F8220D79    cmp     bl, [esi]
.text:F8220D7B    jnz     short loc_F8220D99

```

```

.text:F8220D7D    test    cl, cl
.text:F8220D7F    jz      short loc_F8220D95
.text:F8220D81    mov     bl, [eax+1]
.text:F8220D84    mov     cl, bl
.text:F8220D86    cmp     bl, [esi+1]
.text:F8220D89    jnz     short loc_F8220D99
.text:F8220D8B    add     eax, 2
.text:F8220D8E    add     esi, 2
.text:F8220D91    test    cl, cl
.text:F8220D93    jnz     short loc_F8220D75
.text:F8220D95
.text:F8220D95    loc_F8220D95:                ; CODE XREF: LookupExportedFunction+DFj
.text:F8220D95    xor     eax, eax
.text:F8220D97    jmp     short loc_F8220D9E
.text:F8220D99    ; -----
.text:F8220D99
.text:F8220D99    loc_F8220D99:                ; CODE XREF: LookupExportedFunction+DBj
.text:F8220D99    ; LookupExportedFunction+E9j
.text:F8220D99    sbb     eax, eax
.text:F8220D9B    sbb     eax, 0FFFFFFFh
.text:F8220D9E
.text:F8220D9E    loc_F8220D9E:                ; CODE XREF: LookupExportedFunction+F7j
.text:F8220D9E    test    eax, eax
.text:F8220DA0    jz      short loc_F8220DED
.text:F8220DA2    mov     eax, [esp+30h+var_14]
.text:F8220DA6    mov     ecx, [esp+30h+var_10]
.text:F8220DAA
.text:F8220DAA    loc_F8220DAA:                ; CODE XREF: LookupExportedFunction+CBj
.text:F8220DAA    mov     esi, [esp+30h+var_20]
.text:F8220DAE    inc     ebp
.text:F8220DAF    add     esi, 2
.text:F8220DB2    add     edi, 4
.text:F8220DB5    cmp     ebp, ecx
.text:F8220DB7    mov     [esp+30h+var_20], esi
.text:F8220DBB    jb      short loc_F8220D5C
.text:F8220DBD    mov     ebx, [esp+30h+var_C]
.text:F8220DC1    mov     edi, [esp+30h+var_8]
.text:F8220DC5
.text:F8220DC5    loc_F8220DC5:                ; CODE XREF: LookupExportedFunction+A5j
.text:F8220DC5    ; LookupExportedFunction+B2j
.text:F8220DC5    mov     ecx, [esp+30h+arg_0]
.text:F8220DC9    mov     esi, [esp+30h+var_1C]
.text:F8220DCD    inc     ecx
.text:F8220DCE    add     esi, 4
.text:F8220DD1    cmp     ecx, edi
.text:F8220DD3    mov     [esp+30h+arg_0], ecx
.text:F8220DD7    mov     [esp+30h+var_1C], esi
.text:F8220DDB    jb      loc_F8220D42
.text:F8220DE1
.text:F8220DE1    loc_F8220DE1:                ; CODE XREF: LookupExportedFunction+10j
.text:F8220DE1    ; LookupExportedFunction+21j ...
.text:F8220DE1    xor     eax, eax
.text:F8220DE3
.text:F8220DE3    loc_F8220DE3:                ; CODE XREF: LookupExportedFunction+69j
.text:F8220DE3    ; LookupExportedFunction+162j
.text:F8220DE3    pop     edi
.text:F8220DE4    pop     esi

```

```

.text:F8220DE5    pop     ebp
.text:F8220DE6    pop     ebx
.text:F8220DE7    add     esp, 20h
.text:F8220DEA    retn    0Ch
.text:F8220DED ; -----
.text:F8220DED
.text:F8220DED    loc_F8220DED:                ; CODE XREF: LookupExportedFunction+100j
.text:F8220DED    mov     eax, [esp+30h+var_4]
.text:F8220DF1    mov     ecx, [esp+30h+arg_0]
.text:F8220DF5    lea     ecx, [eax+ecx*4]
.text:F8220DF8    mov     eax, [ecx]
.text:F8220DFA    add     eax, edx
.text:F8220DFC    mov     edx, [esp+30h+arg_8]
.text:F8220E00    test    edx, edx
.text:F8220E02    jz      short loc_F8220DE3
.text:F8220E04    pop     edi
.text:F8220E05    pop     esi
.text:F8220E06    pop     ebp
.text:F8220E07    mov     [edx], ecx
.text:F8220E09    pop     ebx
.text:F8220E0A    add     esp, 20h
.text:F8220E0D    retn    0Ch
.text:F8220E0D    LookupExportedFunction endp

```

User mode calling KAV kernel code directly without a ring 0 transition:

```

kd> bp f824d820
kd> g
Breakpoint 0 hit
klif!sub_F8231820:
001b:f824d820 83ec08      sub     esp,0x8
kd> kv
ChildEBP RetAddr  Args to Child
WARNING: Stack unwind information not available. Following frames may be wrong.
0006f4ec 7432f69c 74320000 00000001 00000000 klif!sub_F8231820
0006f50c 7c9011a7 74320000 00000001 00000000 0x7432f69c
0006f52c 7c91cbab 7432f659 74320000 00000001 ntdll!LdrpCallInitRoutine+0x14
0006f634 7c916178 00000000 c0150008 00000000 ntdll!LdrpRunInitializeRoutines+0x344 (FP0: [Non-Fpo])
0006f8e0 7c9162da 00000000 0007ced0 0006fbd4 ntdll!LdrpLoadDll+0x3e5 (FP0: [Non-Fpo])
0006fb88 7c801bb9 0007ced0 0006fbd4 0006fbb4 ntdll!LdrLoadDll+0x230 (FP0: [Non-Fpo])
0006fc20 f824d749 0106c0f0 0000000e 0107348c 0x7c801bb9
0006fd14 7c918dfa 7c90d625 7c90eacf 00000000 klif!loc_F823173D+0xc
0006fe00 7c910551 000712e8 00000044 0006ff0c ntdll!_LdrpInitialize+0x246 (FP0: [Non-Fpo])
0006fecc 00000000 00072368 00000000 00078c48 ntdll!RtlFreeHeap+0x1e9 (FP0: [Non-Fpo])
kd> t
klif!sub_F8231820+0x3:
001b:f824d823 53          push     ebx
kd> r
eax=0006f3cc ebx=00000000 ecx=00005734 edx=0006f3ea esi=7c882fd3 edi=7432f608
eip=f824d823 esp=0006ef00 ebp=0006f4ec iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
klif!sub_F8231820+0x3:
001b:f824d823 53          push     ebx
kd> dg 1b

                                P Si Gr Pr Lo
Sel   Base   Limit   Type   l ze an es ng Flags

```

```

-----
001B 00000000 ffffffff Code RE    3 Bg Pg P  Nl 00000cfa
kd> !pte eip
      VA f824d823
PDE at  C0300F80      PTE at C03E0934
contains 01010067    contains 06B78065
pfn 1010 ---DA--UWEV    pfn 6b78 ---DA--UREV

```

KAV crashing the system when stepping through its kernel mode code when called from user mode (apparently not that reliable after all!):

```

Breakpoint 0 hit
klif!sub_F8231820:
001b:f824d820 83ec08      sub     esp,0x8
kd> u eip
klif!sub_F8231820:
f824d820 ebfe      jmp     klif!sub_F8231820 (f824d820)
f824d822 085355      or      [ebx+0x55],dl
f824d825 56          push    esi
f824d826 57          push    edi
f824d827 33ed      xor     ebp,ebp
f824d829 6820d824f8 push    0xf824d820
f824d82e 896c2418    mov     [esp+0x18],ebp
f824d832 896c2414    mov     [esp+0x14],ebp
kd> g
Breakpoint 0 hit
klif!sub_F8231820:
001b:f824d820 ebfe      jmp     klif!sub_F8231820 (f824d820)
kd> g
Breakpoint 0 hit
klif!sub_F8231820:
001b:f824d820 ebfe      jmp     klif!sub_F8231820 (f824d820)
kd> bd 0
kd> g
Break instruction exception - code 80000003 (first chance)
*****
*
*   You are seeing this message because you pressed either
*       CTRL+C (if you run kd.exe) or,
*       CTRL+BREAK (if you run WinDBG),
*   on your debugger machine's keyboard.
*
*
*               THIS IS NOT A BUG OR A SYSTEM CRASH
*
*
* If you did not intend to break into the debugger, press the "g" key, then
* press the "Enter" key now. This message might immediately reappear. If it
* does, press "g" and "Enter" again.
*
*****
nt!RtlpBreakWithStatusInstruction:
804e3592 cc          int     3
kd> gu

*** Fatal System Error: 0x000000d1
(0x00003592,0x0000001C,0x00000000,0x00003592)

```

```

Break instruction exception - code 80000003 (first chance)
*****
*
*   You are seeing this message because you pressed either
*       CTRL+C (if you run kd.exe) or,
*       CTRL+BREAK (if you run WinDBG),
*   on your debugger machine's keyboard.
*
*               THIS IS NOT A BUG OR A SYSTEM CRASH
*
* If you did not intend to break into the debugger, press the "g" key, then
* press the "Enter" key now. This message might immediately reappear. If it
* does, press "g" and "Enter" again.
*
*****
nt!RtlpBreakWithStatusInstruction:
804e3592 cc          int      3
kd> g
Break instruction exception - code 80000003 (first chance)

A fatal system error has occurred.
Debugger entered on first try; Bugcheck callbacks have not been invoked.

A fatal system error has occurred.

Connected to Windows XP 2600 x86 compatible target, ptr64 FALSE
Loading Kernel Symbols
.....
Loading User Symbols
.....
Loading unloaded module list
.....
*****
*
*               Bugcheck Analysis
*
*****

Use !analyze -v to get detailed debugging information.

BugCheck D1, {3592, 1c, 0, 3592}

*** ERROR: Module load completed but symbols could not be loaded for klif.sys
Probably caused by : hardware

Followup: MachineOwner
-----
*** Possible invalid call from 804e331f ( nt!KeUpdateSystemTime+0x160 )
*** Expected target 804e358e ( nt!DbgBreakPointWithStatus+0x0 )

nt!RtlpBreakWithStatusInstruction:
804e3592 cc          int      3
kd> !analyze -v
*****
*
*               Bugcheck Analysis
*
*

```

DRIVER_IRQL_NOT_LESS_OR_EQUAL (d1)

An attempt was made to access a pageable (or completely invalid) address at an interrupt request level (IRQL) that is too high. This is usually caused by drivers using improper addresses.
If kernel debugger is available get stack backtrace.

Arguments:

Arg1: 00003592, memory referenced

Arg2: 0000001c, IRQL

Arg3: 00000000, value 0 = read operation, 1 = write operation

Arg4: 00003592, address which referenced memory

Debugging Details:

READ_ADDRESS: 00003592

CURRENT_IRQL: 1c

FAULTING_IP:

+3592

00003592 ?? ???

PROCESS_NAME: winlogon.exe

DEFAULT_BUCKET_ID: INTEL_CPU_MICROCODE_ZERO

BUGCHECK_STR: 0xD1

LAST_CONTROL_TRANSFER: from 804e3324 to 00003592

FAILED_INSTRUCTION_ADDRESS:

+3592

00003592 ?? ???

POSSIBLE_INVALID_CONTROL_TRANSFER: from 804e331f to 804e358e

TRAP_FRAME: f7872ce0 -- (.trap ffffffff7872ce0)

ErrCode = 00000000

eax=00000001 ebx=000275fc ecx=8055122c edx=000003f8 esi=00000005 edi=ddfff298

eip=00003592 esp=f7872d54 ebp=f7872d64 iopl=0 nv up ei pl nz na pe nc

cs=0008 ss=0010 ds=0023 es=0023 fs=0030 gs=0000 efl=00010202

00003592 ?? ???

Resetting default scope

STACK_TEXT:

WARNING: Frame IP not in any known module. Following frames may be wrong.

f7872d50 804e3324 00000001 f7872d00 000000d1 0x3592

f7872d50 f824d820 00000001 f7872d00 000000d1 nt!KeUpdateSystemTime+0x165

0006f4ec 7432f69c 74320000 00000001 00000000 klif+0x22820

0006f50c 7c9011a7 74320000 00000001 00000000 ODBC32!_DllMainCRTStartup+0x52

0006f52c 7c91cbab 7432f659 74320000 00000001 ntdll!LdrpCallInitRoutine+0x14

0006f634 7c916178 00000000 c0150008 00000000 ntdll!LdrpRunInitializeRoutines+0x344

0006f8e0 7c9162da 00000000 0007ced0 0006fbd4 ntdll!LdrpLoadDll+0x3e5

0006fb88 7c801bb9 0007ced0 0006fbd4 0006fbb4 ntdll!LdrLoadDll+0x230

```

0006fbf0 7c801d6e 7ffddc00 00000000 00000000 kernel32!LoadLibraryExW+0x18e
0006fc04 7c801da4 0106c0f0 00000000 00000000 kernel32!LoadLibraryExA+0x1f
0006fc20 f824d749 0106c0f0 0000000e 0107348c kernel32!LoadLibraryA+0x94
00000000 00000000 00000000 00000000 00000000 klif+0x22749

```

```
STACK_COMMAND: .trap 0xffffffff7872ce0 ; kb
```

```
FOLLOWUP_NAME: MachineOwner
```

```
MODULE_NAME: hardware
```

```
IMAGE_NAME: hardware
```

```
DEBUG_FLR_IMAGE_TIMESTAMP: 0
```

```
BUCKET_ID: CPU_CALL_ERROR
```

```
Followup: MachineOwner
```

```

-----
*** Possible invalid call from 804e331f ( nt!KeUpdateSystemTime+0x160 )
*** Expected target 804e358e ( nt!DbgBreakPointWithStatus+0x0 )

```

```

kd> u 804e331f
nt!KeUpdateSystemTime+0x160:
804e331f e86a020000      call     nt!DbgBreakPointWithStatus (804e358e)
804e3324 ebb4             jmp      nt!KeUpdateSystemTime+0x11b (804e32da)
804e3326 90              nop
804e3327 fb             sti
804e3328 8d09             lea      ecx,[ecx]
nt!KeUpdateRunTime:
804e332a a11cf0dfff      mov      eax,[ffdf01c]
804e332f 53              push     ebx
804e3330 ff80c4050000     inc      dword ptr [eax+0x5c4]

```

3.2 The Solution

KAV's anti-virus software relies upon many unsafe kernel-mode hacks that put system stability in jeopardy. Removing unsafe kernel mode hacks like patching non-exported kernel functions or hooking various system services without parameter validation is the first step towards fixing the problem.

Many of the operations where KAV uses hooking or other unsafe means can also be accomplished using documented and safe APIs and conventions that are well-described in the Windows Device Driver Kit (DDK) and Installable File System Kit (IFS Kit). It would behoove the KAV programmers to take the time to read and understand the documented way of doing things in the Windows kernel instead of taking a quite literally hack-and-slash approach that leaves the system at risk of crashes and potentially even privilege escalation.

Many of the unsafe practices relied upon by KAV are blocked by PatchGuard

on x64 and will make it significantly harder to release a 64-bit version of KAV's anti-virus software (which will become increasingly important as computers are sold with x64 support and run x64 Windows by default). Because 32-bit kernel drivers cannot be loaded on 64-bit Windows, KAV will need to port their driver to x64 and deal with PatchGuard. Additionally, assumptions that end user computers will be uniprocessor are fast becoming obsolete, as most new systems sold today support HyperThreading or multiple cores.

Chapter 4

McAfee Internet Security Suite 2006

McAfee's Internet Security Suite 2006 package includes a number of programs, including anti-virus, firewall, and anti-spam software. In particular, however, this article discusses one facet of Internet Security Suite 2006: The McAfee Privacy Service.

This component is designed to intercept outbound traffic and sanitize it of any predefined sensitive information before it hits the wire. This chapter will give brief background on some of the technology that McAfee uses and how their implementation may lead to problems.

4.1 The Problem

From the very start, if one is familiar with network programming, such a goal would appear to be very difficult to practically achieve. For instance, many programs send data in a compressed or encrypted form, and there is no common way to process such data without writing specialized software for each target application. This immediately limits the effectiveness of the Privacy Service software's generalized information sanitization process to programs that have a) had specialized handler code written for them, or b) send information to the Internet in plaintext. Furthermore, the very act of modifying an outbound data stream could potentially cause an application to fail (consider the case where an application network protocol includes its own checksums of data sent and received, where arbitrary modifications of network traffic might cause it to be rejected).

The problem with McAfee Internet Security Suite goes deeper, however. The mechanism by which Internet Security Suite intercepts (and potentially alters) outbound network traffic is through a Windows-specific mechanism known as an LSP (or Layered Service Provider).

LSPs are user mode DLLs that "plug-in" to Winsock (the Windows sockets API) and are called for every sockets API call that a user mode program makes. This allows easy access to view (and modify) network traffic without going through the complexities of writing a conventional kernel driver. An LSP is loaded and called in the context of the program making the original socket API call.

This means that for most programs using user mode socket calls, all API calls will be redirected through the Internet Security Suite's LSP, for potential modification.

If one has been paying attention so far, this approach should already be setting off alarms. One serious problem with this approach is that since the LSP DLL itself resides in the same address space (and thus has the same privileges) as the calling program, there is nothing technically stopping a malicious program from modifying the LSP DLL's code to exempt itself from alteration, or even bypassing the LSP directly.

Unfortunately, the flaws in the McAfee Privacy Service do not simply end here. Already the technical limitations of an LSP for securely intercepting and modifying network traffic make this approach (in the author's opinion) wholly unsuitable for a program designed to protect a user from having his or her private data stolen by malicious software.

Specifically, there are implementation flaws in how the LSP itself handles certain socket API calls that may cause otherwise perfectly working software to fail when run under McAfee Internet Security Suite 2006. This poses a serious problem to software vendors, who are often forced to interoperate with pervasive personal security software (such as Internet Security Suite).

The Windows Sockets environment is fully multithreaded and thread-safe, and allows programs to call into the sockets API from multiple threads concurrently without risk of data corruption or other instability. Unfortunately, the LSP provided by McAfee for its Privacy Service software breaks this particular portion of the Windows Sockets API contract. In particular, McAfee's LSP does not correctly synchronize access to internal data structures when sockets are created or destroyed, often leading to situations where a newly created socket handed back to an application program is already mistakenly closed by the flawed LSP before the application even sees it.

In addition, the author has also observed a similar synchronization problem regarding the implementation of the 'select' function in the Privacy Service LSP. The select function is used to poll a set of sockets for a series of events, such as data being available to read, or buffer space being available to send data. The

McAfee LSP appears to fail when calls to select are made from multiple threads concurrently, however. It often appears to switch a socket handle specified by the original application program with an entirely different handle. In Windows, the same handle space is shared by socket handles and all other types of kernel objects, such as files or processes and threads. This subsequently results in calls to select failing in strange ways, or worse, returning that data is available for a particular socket when it was in fact available on a different socket entirely.

Both of these flaws result in intermittent failures of correctly written third party applications when used in conjunction with McAfee Internet Security Suite 2006.

4.2 The Solution

If one is stuck in the unfortunate position of being forced to support software running under McAfee Internet Security Suite 2006, one potential solution to this problem is to manually serialize all calls to select (and other functions that create or destroy sockets, such as socket and the WSASocket family of functions). This approach has worked in practice, and is perhaps the least invasive solution to the flawed LSP.

An alternative solution is to bypass the LSP entirely and instead call directly to the kernel sockets driver (AFD.sys). However, this entails determining the actual handle associated with a socket (the handle returned by the McAfee LSP is in fact not the underlying socket handle), as well as relying on the as of yet officially undocumented AFD IOCTL interface.

From McAfee's perspective, the solution is fairly simple: correctly serialize access to internal data structures from function calls that are made from multiple threads concurrently.

Chapter 5

Conclusion

As the Internet becomes an increasingly hostile place and the need for in-depth personal security software (as a supplement or even replacement for proper system administrator) grows for end-users, it will become increasingly important for the vendors and providers of personal security software to ensure that their programs do not impair the normal operation of the systems upon which their software is installed. The author realizes that this is a very difficult task given what is expected of most personal security software suites, and hopes that by shedding light on the flaws in existing software, new programs can be made to avoid similar mistakes.