A reversed approach to security - Deeper malware analysis (practical session)

Alexandre Dulaunoy

8th February 2005

Contents


```
"- After everything that has happened, how can you expect me to believe you?"
"- I don't. I expect what I always expected, for you to make up your
own damn mind. Believe me or don't."

The Matrix - The Matrix Reloaded ("Matrix 2")
```

1 Introduction to Malware analysis

After the "basic" network analysis of part 4, you have now a better understanding of what happened in the network. Not everything is clear (it's impossible to reach the truth) but you are facing various part of the attacks and you would like to dig into the components used during the attacks. Some components seem to be software but how to make analysis ?

1.1 Analysis approach

There are two ways to analyze unknown components on a compromised system. The dynamic analysis approach and the static analysis approach. Static analysis is the most difficult way and takes a lot time to be properly done but the results can be very good. On the other side, the dynamic analysis is often faster to realize and give sometimes sufficient results.

1.1.1 Dynamic analysis

Dynamic analysis means that you are analyzing software during its execution on a computer or in an virtual machine.

- "Black box approach". You are studying the component without digging the internal but only the flows (input/output) of the program, its interactions with the external interface (read/write access to files) or its effects on the environment (e.g. smart-card and power usage).
- "Post mortem approach". You are studying the effect on the system after the execution of the component. Effects can be memory effects, file access, temporary data created in the file system,...
- "Conventional approach". You are studying the execution of the component actively with system call tracker, memory analyzer, real time debugger/wrapper, open files tracker, ...

Please note that dynamic analysis can be dangerous if not properly executed. You have been warned...

1.1.2 Static analysis

- "Classical static analysis" is the method to look at the component without any execution on the component itself. You can look a the string contained in the binary, compares the hashes fingerprint of the software,
- "Disassembly analysis" is to recover the machine-language code of a component. This can be a complex and long task to analyze large software but realivility small software can be disassembled.
- "Decompilation analysis" is to recover the source code of the component from machine-language. Sometimes compiler (from source code to machine-language) adds a lot of information in the compiled program. (e.g. think about

1.2 Practical analysis on your own "malware"

Before starting with dangerous components, you can start simply with a very small trivial C and bad written program.

```
int
main (argc, argv) int argc; char ** argv;
{
        char val[16] ;
        strcpy(val, argv[1]);
}
```

We invite you to compile the program with a C compiler.

You can start to evaluate the program with the following tools and approach :

1.2.1 Static Analysis

- strings
- nm
- objdump
- others ?

1.2.2 Dynamic Analysis

- executing the program
- executing + call tracking (from strace to lsof)
- executing within a debugger (from gdb to ddd)
- others ?

1.3 Pratical analysis on the attack(s)

You must establish a strategy in order to analyze the attacker(s)'s code.

1.4 Shell code

A basic shell code example for a version of Linux on i386.

```
 0xeb, 0x18,       : jmp  +0x18           : jump L2
L1:
 0x5e,             : pop  %esi            : %esi = L3
 0x89, 0x76, 0x08, : mov  %esi,0x08(%esi) : esi+[8..11] = %esi
```

```
                                                L3+[8..11] = %esi
 0x31, 0xc0,       : xor  %eax,%eax       : %eax = 0
 0x88, 0x46, 0x07, : mov  %al,0x07(%esi) : esi+7 = 0
                                                L3+7 = 0
 0x89, 0x46, 0x0c, : mov  %eax,0x0c(%esi) : esi+[12..15] = 0
                                                L3+[12..15] = 0
 0xb0, 0x0b,       : mov  $0x0b,%al       : %eax = 11 (execve)
 0x89, 0xf3,       : mov  %esi,%ebx       : %ebx = L3
 0x8d, 0x4e, 0x08, : lea  0x08(%esi),%ecx : %ecx = L3+8
 0x8d, 0x56, 0x0c, : lea  0x0c(%esi),%edx : %edx = L3+12
 0xcd, 0x80,       : int  $0x80           : syscall
L2:
 0xe8, 0xe3, 0xff,
 0xff, 0xff,       : call -0x18           : call L1
L3:
 '/', 'b', 'i', 'n', '/', 's', 'h',       : "/bin/sh"
 0x00
```

2 Bibliography

- Smashing the Stack for Fun and Profit, Aleph One, Phrak 49, 1996.
- Forensic Discovery, Dan Farmer, Wietse Venema, Addison Wesley, 2005
- GNU tools man page.