Module 6.1

Creating undetectable applications

Understanding how antiviruses detect threats

Antivirus (AV) detection of threats makes use of three key techniques:

1. Static analysis in the search of signatures,
2. Emulating code in the search of suspicious activity,
3. Heuristics.

Signatures

Let's start with an overview of signature-based code analysis. A malware signature is a string characteristic of a suspicious program an antivirus searches for in a scanned file. Antivirus vendors are constantly working to analyze suspicious files (including files sent to AV labs). They extract strings typical for given software and upload them to be sent to users' AV databases when they are updated. To make a program undetectable, we usually begin with encrypting as much code as we can, so that an antivirus fails to identify malware signatures in it. Still, the decryption procedure is not encrypted. This means that this anti-discovery solution should be used last.

Our task is to find a signature and replace it with a code excerpt that does the exact same thing but uses different commands. We can also change the order of instructions within the signature (provided it doesn't change the way the program operates). We'll use `Dsplit` for this purpose. The program splits analyzed programs into chunks, with each subsequent output part having more code than the previous one. This makes it easy to quickly find the signature that sets off AV alarms.

The program runs in the console. Here's the syntax:

```
Dsplit.exe [start] [end] [incrementation_of_subsequent_parts] [file.exe]
```

The arguments obviously should be given without the square brackets. Here's an example of usage:

```
Dsplit.exe 0 max 1000 virus.exe
```

Assuming `virus.exe` is 10,000 bytes, the output is 10 files with the respective sizes 1,000, 2,000, 3,000 … 10,000 bytes. We scan them all and look at the scan results to see which one is detected and has the smallest number. Let's assume this is the 3,000 file. What can be gathered is that the signature is located between the 2,000 and 3,000 byte in the file. We run `Dsplit` again, this time reducing the scan area.

```
Dsplit.exe 2000 3000 100 virus.exe
```

The output is again 10 files different in size, only this time the difference is smaller. We repeat the action until the exact position of the signature is identified. Next, we convert this address to a hexadecimal format (you can do it on paper or use the system calculator).

After the conversion, we open the file in `LordPE`. Using the `FLC` option, we type the calculated value into `Offset` and click `Calculate`. The result is the `VA` and `RVA` address in the file. We'll also discover the section the found address points to. If it's a code section, we open the program in a debugger and jump to this location. At this point, you can freely change any code you need. If there's data in the section, it also needs to be encrypted.

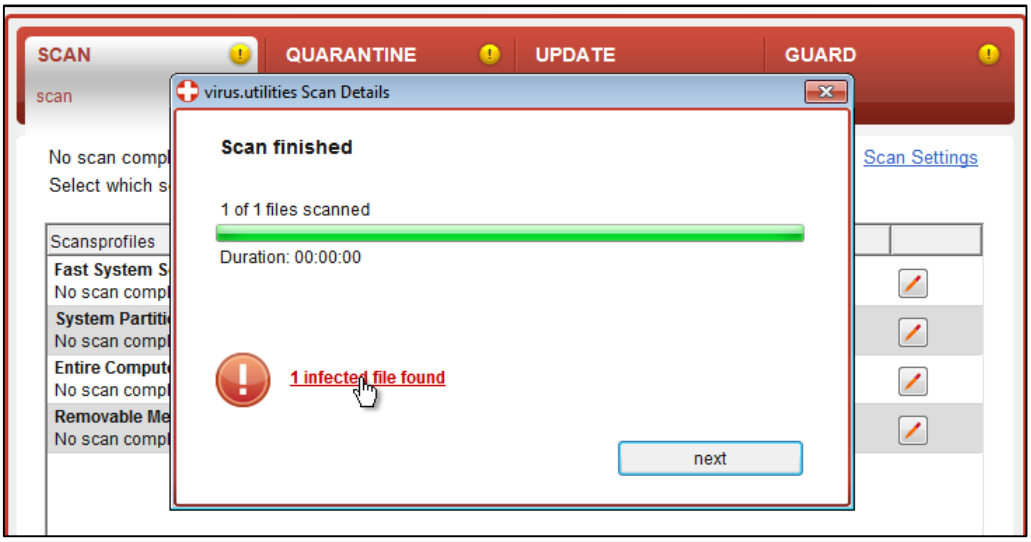
Practice: video module transcript

Welcome to the sixth module of this training. Through the course of this module, we'll learn how to write undetectable applications and make the program invisible for anti-virus software. As our first move, let’s consider which techniques are used by anti-virus software to detect threats. First of all, anti-virus programs make use of signatures and that's our subject in this part of the module. They also use heuristics, which we'll thoroughly discuss in the next part. Finally, anti-virus programs also use emulation, and I'll tell you more about it in the third part of this module.

Use of signatures is definitely the simplest way of detecting malicious software used by anti-virus programs. They are simply byte strings which are characteristic for the given program. We get them when updating our anti-virus. But how do we seek signatures?

For this task, we'll use the Dsplit program to split our application into parts, each larger than the previous one. Thanks to this, we'll be able to figure from which byte we’ve been detected. The moment we find the signature and specify its location, we have to determine the section in which it's present. If the signature is present in the code section, we'll edit it using Olly debugger, and if in the data section, we'll use hexplorer.

Let's see how to search for signatures in practice. Here we have the reshacker program. It's a popular resource editor. It's not dangerous for us, but was encoded or packed using a packer recognized by anti-virus software as potentially dangerous. We won't use programs from the previous parts, because they weren't added to the anti-virus databases and won't activate the alarm.

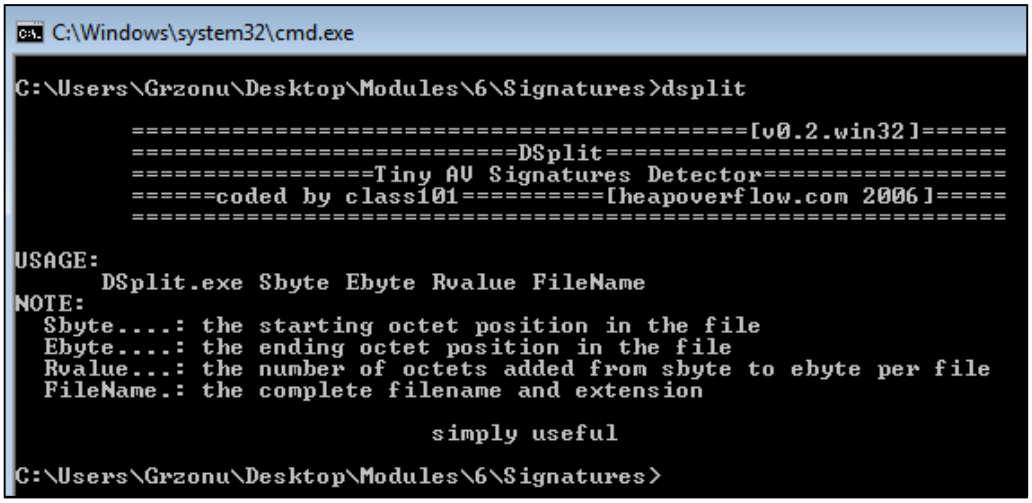


Now let's scan the reshacker program using the Ikarus anti-virus. We can see that it detected a threat. Additionally, we'll send our program to the VirusTotal online service, which will scan the application using a few dozen scanners. The file has loaded to the server. Scanning is currently in progress.

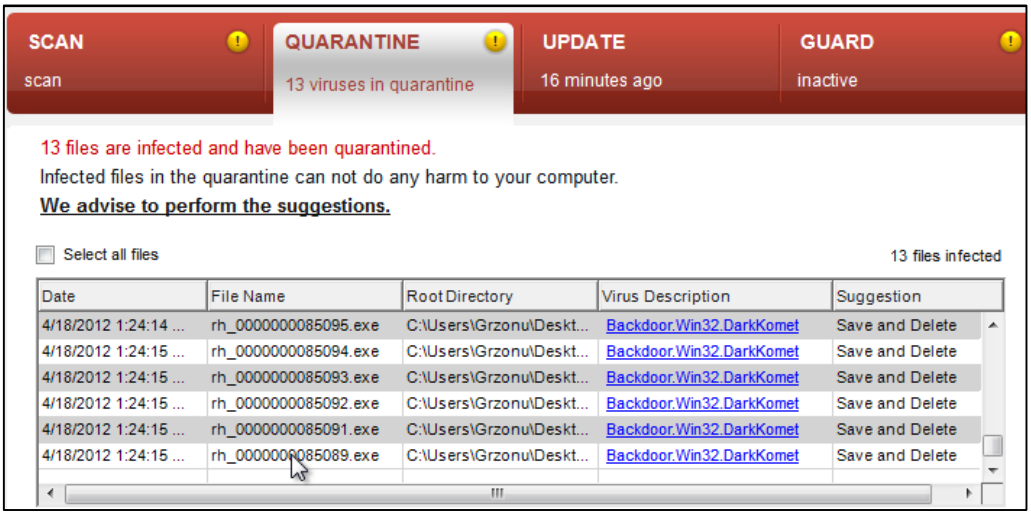
Comodo	-
DrWeb	-
Emsisoft	Backdoor.Win32.DarkKomet!IK
eSafe	-
eTrust-Vet	-
F-Prot	-
F-Secure	-
Fortinet	-
GData	-
Ikarus	Backdoor.Win32.DarkKomet

As we can see, the thread was detected by the programs Emisoft and ikarus. The analysis is still not over, but as for now, let's deal with modifying the

program so that it's undetectable by the ikarus. Let's see how it should be done. We use the Dsplit program to divide the file into parts.



It takes the following parameters: the start of the section, the end of the section, the value by which the files will be increased, and the name. Each part will be larger than the previous one by 100.000 bytes. We can see that the program created the relevant files. We got as many as 13 parts, let's scan them independently. All 13 files were detected, so the signature has to be present before the byte no. 100.000. We remove the remaining files and split our program from 0 to 100.000 bytes, but this time each 10.000 bytes. Let's scan it again.



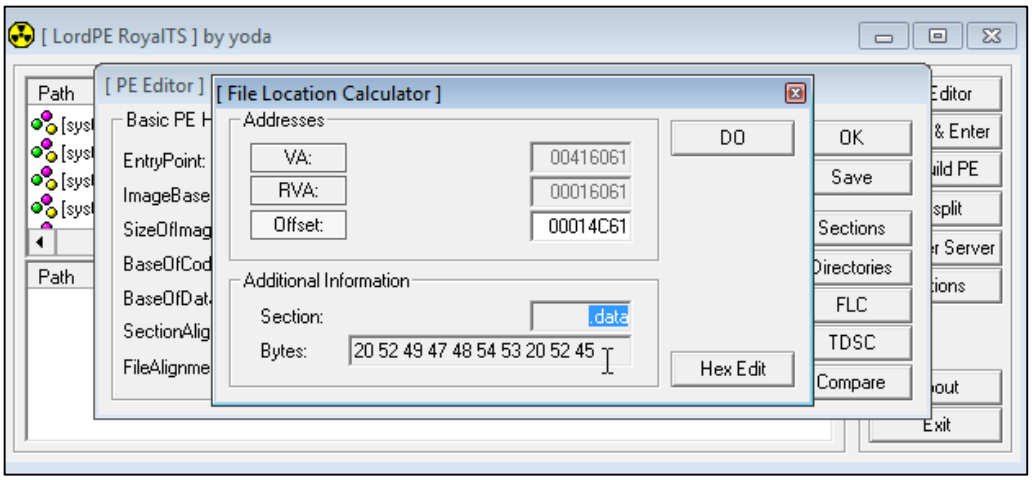
We can see that the smallest detected file is the file that ends with 90.000, so the signature that activates the anti-virus software is present somewhere between byte no. 80.000 and 90.000. We write: 80.000, 90.000 and this time increase each 1.000 bytes. Let's scan it again. As we can see, the signature is present somewhere before the byte no. 86.000. If so, we remove the files and this time we split the section between 85.000 and 86.000 to be separated each 100 bytes.

This time all 10 files were detected, so the signature is present somewhere before byte no. 85.100. Now we can set the section from 84.900 to 86.000 with the interval of 10 bytes. Unfortunately, we've set a bit too much. Let’s remove these parts and set 84.900 to 86.000, each 100 bytes. Again, the youngest file is 85.100, so the signature is between the byte 84.900 and 85.100. This tells us a lot more. Now we'll set this section and set the interval to 10 bytes.

```
C:\Users\Grzonu\Desktop\Modules\6\Signatures>dsplit 0 100000 10000 rh.exe

        ============================================[v0.2.win32]======
        ==========================DSplit============================
        ================Tiny AV Signatures Detector=================
        ======coded by class101==========[heapoverflow.com 2006]=====
        ==================================================================


===[ Analyzation ]====================
[-passed-] accessing the file
[-passed-] buffering the content
[] file size: 1273024
[] work size: 100000
[] sbyte: 0
[] ebyte: 100000

===[ Files Creation ]=================
[-passed-] creating the files [100%]
[] files: 10
```

The result is 85.090, so the signature is present somewhere before this byte. Now we set the difference to 1 byte. We see that 12 files were detected. We've figured out the relevant address, it's the byte no. 85089. The signature was detected under this address. We remove all files apart from the one with the address of our signature and use the LordPE program to find the place specified by the address. But first, we use a calculator to convert the address of our signature to the hexadecimal system. It's 14C61.

Now let's use the FLC option from LordPE to find the right place. We enter our address in the offset field. As we can see, it's a data section. It's not a code, so we'll use hexplorer to edit it. Now we copy this value and drag the rh.exe file to hexplorer. We use the option Go to address and enter 14C61 in the offset field.



It seems that it points to a character string, so we can freely remove it from the file. We select the area and choose Reset selection. We see that everything was reset. We save the new program as rh2.exe. Now we scan it using a local anti-virus software. As we can see, no infection was found. Let's send it to the VirusTotal website to check whether nothing gets detected.

The file is loaded to the server. Let's wait for the scanning results. By the way, we can see that emisoft stopped detecting the virus. Let's wait for the analysis to complete, meanwhile we'll check whether the program still works after our modification.

We start the new application version and, as we can see, the program still works well after the introduced modification. The analysis is over, and the result is stunning. None of the 42 anti-virus programs considered our program a threat.

To summarize this part of the module, we've learnt how to relatively quickly find the bytes which belong to the signature activating the anti-virus alarm, and also how to modify the program once it's detected. In the next parts of this module we'll learn how to deal with other (definitely much more advanced) detection methods, the already-mentioned heuristics and emulation. Thanks for your attention and see you in the next part.