<u>CSI4139</u>
Lab 3
NamChi Nguyen - 7236760

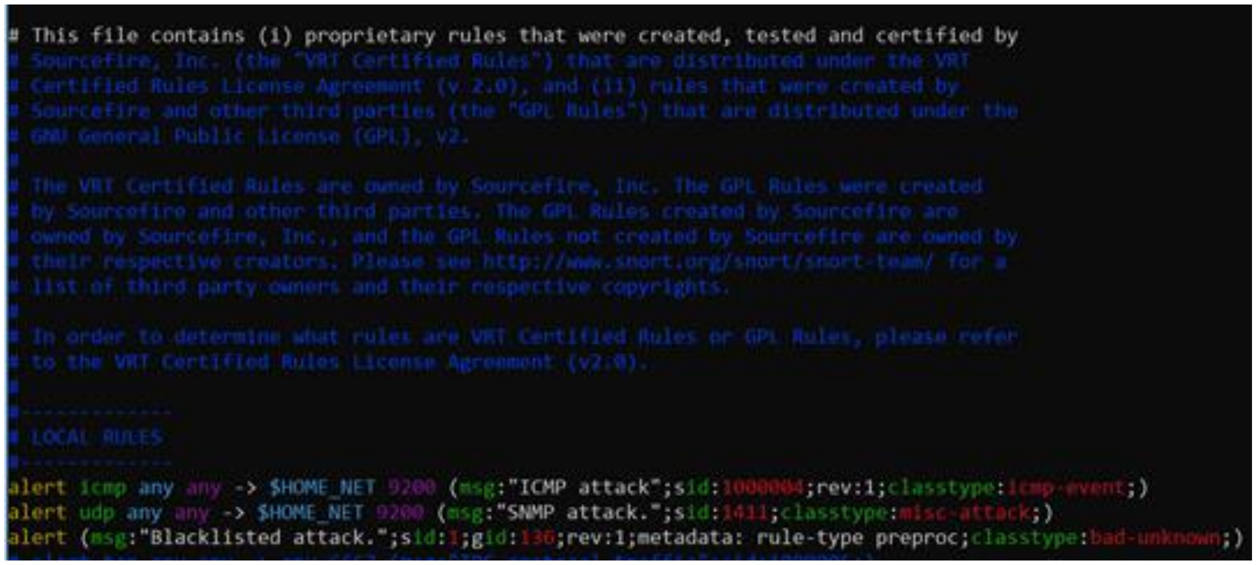
<u>Group members:</u>
Julian Templeton - 8229400
Sean McAvoy - 8263210

Part A: Snort rules

<u>Note</u>: We were unable to successfully compile Snort within Docker using the VM to simulate the attacks to test the rules. However, we will discuss the attacks that we would've tested on these rules.

Hence, the rules based on the 3 attacks are as follows:

- $HOME_NET is the destination IP address that would be set in the Snort configuration file.



```
# This file contains (i) proprietary rules that were created, tested and certified by
# Sourcefire, Inc. (the "VRT Certified Rules") that are distributed under the VRT
# Certified Rules License Agreement (v 2.0), and (ii) rules that were created by
# Sourcefire and other third parties (the "GPL Rules") that are distributed under the
# GNU General Public License (GPL), v2.
#
# The VRT Certified Rules are owned by Sourcefire, Inc. The GPL Rules were created
# by Sourcefire and other third parties. The GPL Rules created by Sourcefire are
# owned by Sourcefire, Inc., and the GPL Rules not created by Sourcefire are owned by
# their respective creators. Please see http://www.snort.org/snort/snort-team/ for a
# list of third party owners and their respective copyrights.
#
# In order to determine what rules are VRT Certified Rules or GPL Rules, please refer
# to the VRT Certified Rules License Agreement (v2.0).
#
#-------------
# LOCAL RULES
#-------------
alert icmp any any -> $HOME_NET 9200 (msg:"ICMP attack";sid:1000004;rev:1;classtype:icmp-event;)
alert udp any any -> $HOME_NET 9200 (msg:"SNMP attack.";sid:1411;classtype:misc-attack;)
alert (msg:"Blacklisted attack.";sid:1;gid:136;rev:1;metadata: rule-type preproc;classtype:bad-unknown;)
```

Figure 1. Local rules

ICMP-based attack [1], [2]*:*
alert icmp any any -> $HOME_NET any (msg:"ICMP attack"; sid:1000004; rev:1; classtype:icmp-event;)

- This type of attack would've been a ping flood in which the rule would produce an alert of the numerous amounts of pings sent to $HOME_NET on any destination port from any source IP address and source port.
- sid:1000004 represents the Snort ID that identifies this rule in which any number $\geq$ 1,000,000 is used for local rules.
- rev:1 represents the revision number of Snort rules alongside its sid.
- classtype:icmp-event represents the category of the rule that'll detect an attack that's a part of a generalized class of icmp events.
- A limitation to this rule is that no restriction on the number of pings within a short period of time is specified in order to distinguish between normal and suspicious ping activity.

SNMP-based attack [2], [3]:
<u>Note</u>: This rule is referenced from the Snort community rules.

```
alert udp any any -> $HOME_NET any (msg:"SNMP attack"; sid:1411; classtype:misc-attack)
```

- This type of attack would've been a flood of UDP packets in which the rule would produce an alert of the numerous amounts of UDP packets sent to $HOME_NET on any destination port from any source IP address and source port.
- sid:1411 represents the Snort ID that identifies this rule in which any number from 100 to 999,999 is a rule included in the Snort distribution.
- classtype:misc-attack represents the category of the rule that'll detect an attack that's a part of a generalized class of miscellaneous attacks.

Attack from blacklisted source [2], [4]*:*

```
alert (msg: "Blacklisted attack"; sid:1; gid:136; rev:1; metadata: rule-type preproc; classtype:bad-
unknown;)
```

- This type of attack would've been from any malicious IP address listed in the blacklist rules file in which if any of the source or destination IP addresses match the blacklist file then an alert is produced and the preprocessor halts further action [4]. This rule could help to prevent malicious hosts from gaining access to the network.
- sid:1 represents the Snort ID that identifies this rule in which any number < 100 is reserved and 1 is reserved for packets that are blacklisted.
- gid:136 represents the generator ID that identifies the part of Snort that produces an event when a specific rule is triggered by the reputation preprocessor.
- classtype:bad-unknown represents the category of the rule that'll detect an attack that's a part of a generalized class of potentially bad traffic.
- A limitation to this rule is that since Snort would run in IDS mode, the malicious packets would not be blocked/dropped but it would occur if we ran Snort in NIPS (network intrusion prevention system) mode [4].

To conclude, after working with the Snort IDS and learning how it generates alerts for suspicious activity and monitors network traffic, an ideal intrusion detection system should also be able to contain the attack, minimize the amount of damage caused as well as block further attacks and allow the system to recover from a non-secure state to a secure state.

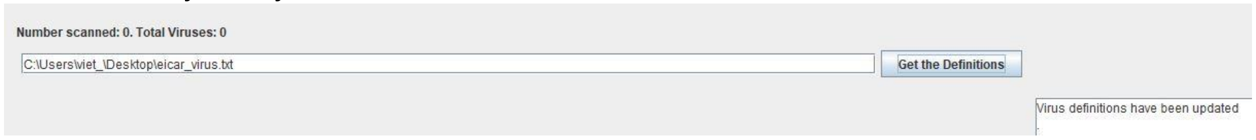
Part B: Antivirus program
Our antivirus program was implemented using the programming language Java in a Windows OS environment.

Assumptions for the program to run:

- We used the EICAR virus file provided in the lab: X5O!P%@AP[4\\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H* [5]. However, we encrypted the virus using the Caesar cipher. The definition file would essentially contain all possible encryptions of the EICAR virus, although we encrypted with a shift of 3 and used that as our sample virus signature for testing purposes: A5R!S%@DS[4\SCA54(S^)7FF)7}$HLFDU-VWDQGDUG-DQWLYLUXV-WHVW-ILOH!$K+K*.
- The files that are scanned for viruses are .txt files.
- If the quarantined folder doesn't exist, a new one is created to isolate the files.

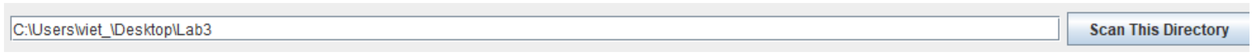
<u>Program flow:</u>
First, the program reads the virus definition file (ie. a .txt file) and creates a list of virus signatures. Then, the user must provide a path to the directory that they wish to scan for viruses (which initially doesn't contain a quarantined folder).

Path to read definition file:

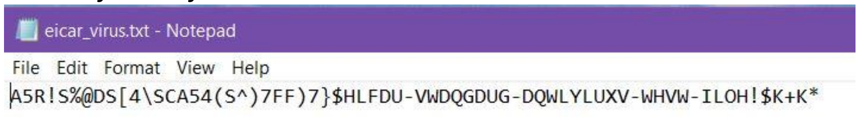

Number scanned: 0. Total Viruses: 0
C:\Users\viet_\Desktop\eicar_virus.txt

Virus definitions have been updated

Path to scan:

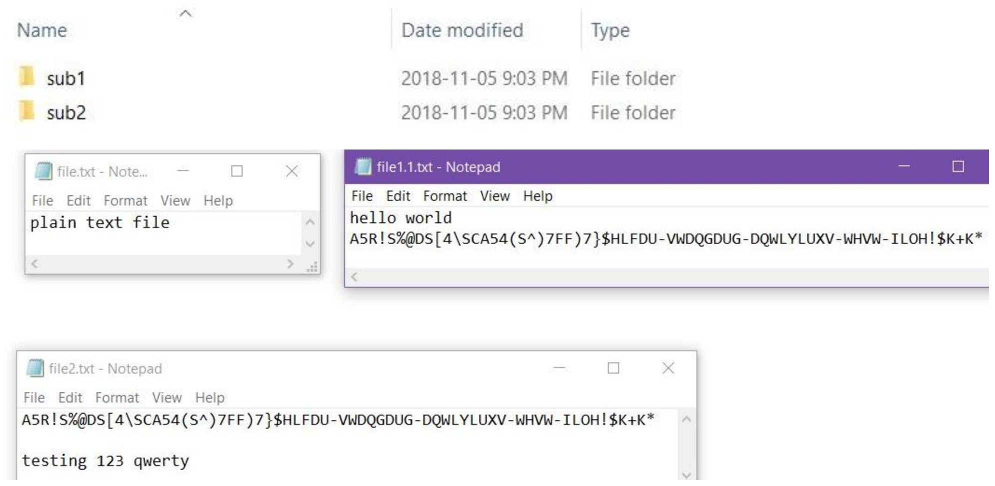

C:\Users\viet_\Desktop\Lab3

Virus definition file:


eicar_virus.txt - Notepad

```
A5R!S%@DS[4\SCA54(S^)7FF)7}$HLFDU-VWDQGDUG-DQWLYLUXV-WHVW-ILOH!$K+K*
```

Directory to scan and its files:



Name	Date modified	Type
sub1	2018-11-05 9:03 PM	File folder
sub2	2018-11-05 9:03 PM	File folder

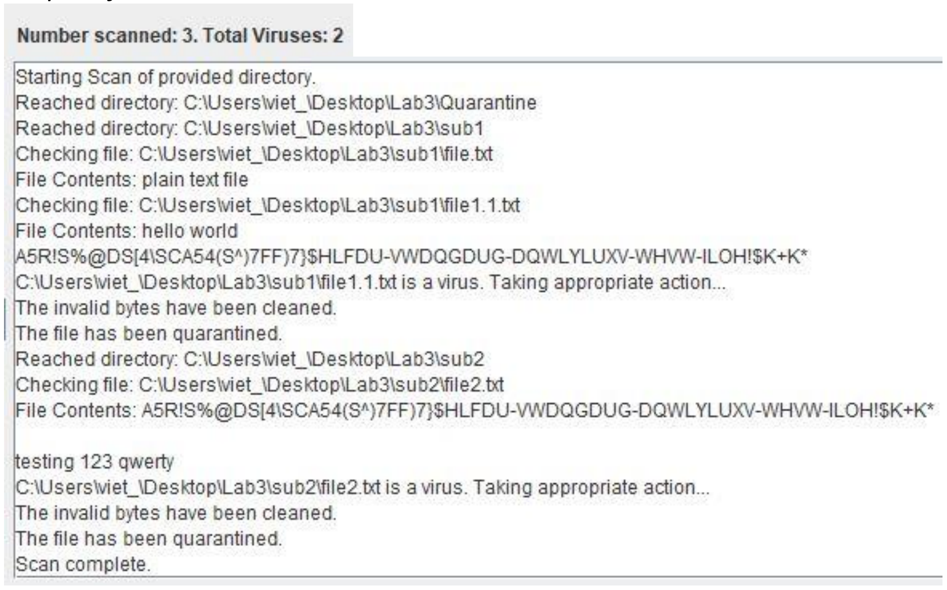
file.txt - Note...
```
plain text file
```

file1.1.txt - Notepad
```
hello world
A5R!S%@DS[4\SCA54(S^)7FF)7}$HLFDU-VWDQGDUG-DQWLYLUXV-WHVW-ILOH!$K+K*
```

file2.txt - Notepad
```
A5R!S%@DS[4\SCA54(S^)7FF)7}$HLFDU-VWDQGDUG-DQWLYLUXV-WHVW-ILOH!$K+K*

testing 123 qwerty
```

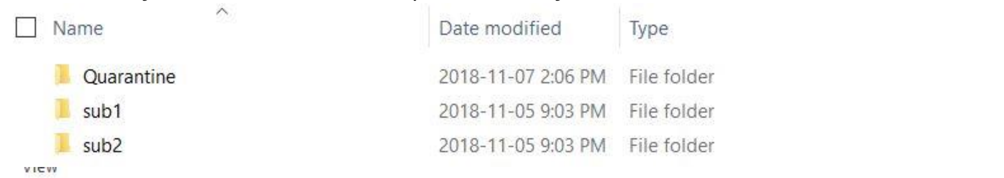
Next, the directory and its subdirectories are scanned recursively to search for infected files. Since the virus signature is assumed to be polymorphic, a Caesar cipher must be done to obtain all possible forms of the virus signature and then compare with the signature found in the definition file. When an infected file is found, it is modified, and the first 8 bytes of the virus signature is replaced with “xxxxxxxx”. Afterwards, the now quarantined files are moved to a new folder called Quarantine that has been created to isolate the files. The number of files scanned, and a counter of the viruses found are also included.

Output of scan:

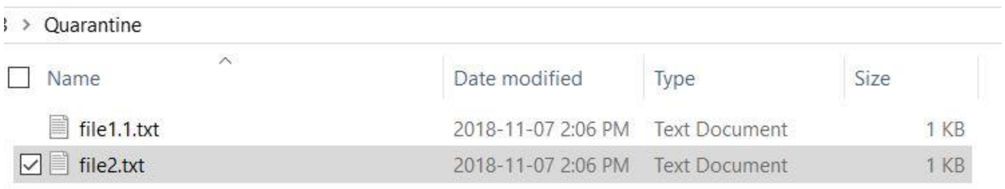


```
Number scanned: 3. Total Viruses: 2

Starting Scan of provided directory.
Reached directory: C:\Users\viet_\Desktop\Lab3\Quarantine
Reached directory: C:\Users\viet_\Desktop\Lab3\sub1
Checking file: C:\Users\viet_\Desktop\Lab3\sub1\file.txt
File Contents: plain text file
Checking file: C:\Users\viet_\Desktop\Lab3\sub1\file1.1.txt
File Contents: hello world
A5R!S%@DS[4\SCA54(S^)7FF)7}$HLFDU-VWDQGDUG-DQWLYLUXV-WHVW-ILOH!$K+K*
C:\Users\viet_\Desktop\Lab3\sub1\file1.1.txt is a virus. Taking appropriate action...
The invalid bytes have been cleaned.
The file has been quarantined.
Reached directory: C:\Users\viet_\Desktop\Lab3\sub2
Checking file: C:\Users\viet_\Desktop\Lab3\sub2\file2.txt
File Contents: A5R!S%@DS[4\SCA54(S^)7FF)7}$HLFDU-VWDQGDUG-DQWLYLUXV-WHVW-ILOH!$K+K*

testing 123 qwerty
C:\Users\viet_\Desktop\Lab3\sub2\file2.txt is a virus. Taking appropriate action...
The invalid bytes have been cleaned.
The file has been quarantined.
Scan complete.
```

Quarantine folder created and its quarantined files:

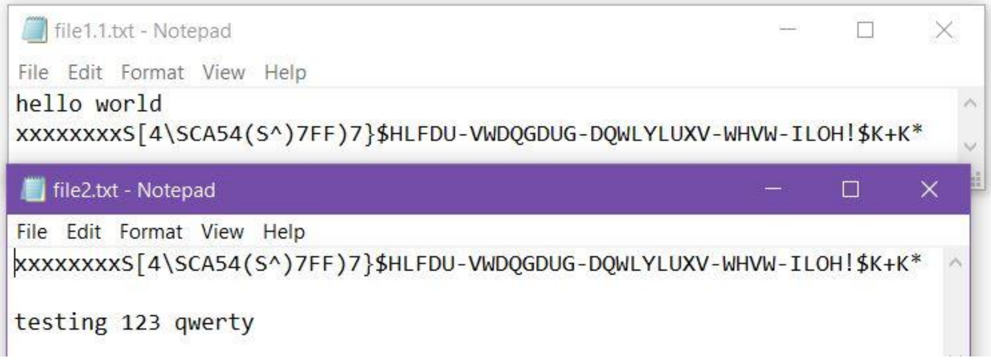


Name	Date modified	Type
Quarantine	2018-11-07 2:06 PM	File folder
sub1	2018-11-05 9:03 PM	File folder
sub2	2018-11-05 9:03 PM	File folder



› Quarantine

Name	Date modified	Type	Size
file1.1.txt	2018-11-07 2:06 PM	Text Document	1 KB
file2.txt	2018-11-07 2:06 PM	Text Document	1 KB



file1.1.txt - Notepad

```
hello world
xxxxxxxxS[4\SCA54(S^)7FF)7}$HLFDU-VWDQGDUG-DQWLYLUXV-WHVW-ILOH!$K+K*
```

file2.txt - Notepad

```
xxxxxxxxS[4\SCA54(S^)7FF)7}$HLFDU-VWDQGDUG-DQWLYLUXV-WHVW-ILOH!$K+K*

testing 123 qwerty
```

Production environment:

Since our program is a rudimentary antivirus scanner, there are many additional functionalities that are required for it to be useful in a production environment.

Here are a few examples [6]:

- In our program, a local quarantined folder is created to isolate infected files, however a new folder shouldn't be created each time the scan is executed. It should be a centralized folder instead to store all infected files.

- Our program doesn't automatically delete quarantined files, so the scanner should be able to remove files immediately or after a certain period of time without the user having to manually delete the files.

- Real-time scanning should be added to scan all directories without the user manually choosing a directory to scan so that malware doesn't go undetected.

- The scanner should also have automatic updates in order to have the latest virus definitions and threats that arise from new malware that a user's computer can be vulnerable to if the scanner is not up-to-date.

To conclude, there are countless other functionalities that are required in a production antivirus software to protect a system from malware. It demonstrates how crucial and diligent AV software must be in order to defend against malicious entities and keep systems secure.

References:

[1] Infosec Institute, “Basic Snort Rules Syntax and Usage”, n.d., [Online]. Available: https://resources.infosecinstitute.com/snort-rules-workshop-part-one/. [Accessed: Oct. 29, 2018].

[2] Snort, “Snort Users Manual 2.9.12”, Jun. 13, 2018, [Online]. Available: http://manual-snort-org.s3-website-us-east-1.amazonaws.com/. [Accessed: Oct. 29, 2018].

[3] Snort, “Downloads”, n.d., [Online]. Available: https://www.snort.org/downloads#rules. [Accessed: Oct. 29, 2018].

[4] N. Dietrich, “The Reputation Preprocessor in Snort – Blacklists and Whitelists”, Dec. 10, 2015, [Online]. Available: https://sublimerobots.com/2015/12/the-snort-reputation-preprocessor/. [Accessed: Nov. 10, 2018].

[5] European Expert Group For IT Security, “Intended Use”, n.d., [Online]. Available: http://www.eicar.org/86-0-Intended-use.html. [Accessed: Oct. 31, 2018].

[6] B. Martin, “5 Must-Have Features of Antivirus Software”, Jul. 21, 2016, [Online]. Available: https://www.greatlakescomputer.com/blog/5-must-have-features-of-antivirus-software. [Accessed: Nov. 7, 2018].

Lab code:

[1] GeeksforGeeks, “Java program to List all files in a directory and nested sub-directories | Recursive approach”, n.d., [Online]. Available: https://www.geeksforgeeks.org/java-program-list-files-directory-nested-sub-directories-recursive-approach/. [Accessed: Nov. 1, 2018].

[2] Stackoverflow, “Stack swing elements from top to bottom”, Mar. 12, 2012, [Online]. Available: https://stackoverflow.com/questions/9674774/stack-swing-elements-from-top-to-bottom. [Accessed: Nov. 5, 2018].