HY-457: Introduction to Information Security Systems

Computer Science Department

Spring Semester 2024

Assignment 2

“Implementation of a Ransomware Protection Software Suite”

Tutorial: 21/03/2024
Deadline: 21/04/2024

Introduction

Over the past few weeks, multiple reports have emerged about the work of KozaliBear, an infamous group of cybercriminals that are known for breaking into corporate networks and organizations’ cloud environments. This group is known to use viruses to infect the workstations of unsuspected employees, gain unauthorized access to classified data and even steal proprietary information. What makes this group extremely dangerous is the use of ransomware. Ransomware is a type of malware that locks the user’s personal files until a ransom is paid [1]. Files are encrypted and the only way to get the decryption key is if the victim pays a specified amount of money. The KozaliBear group has managed to extort millions of money using ransomware. Finally, they usually ask for money in the form of cryptocurrencies (e.g. Bitcoin) [2] since they are not trivial (but not impossible) to trace.

You are a cybersecurity engineer working for a leading manufacturer of automotive electronics. Last night, your supervisor was informed that one of your competitors was attacked by KozaliBear, and a ransomware was installed into all of their systems making them practically unusable. Your supervisor is concerned that you might be the next target or that some of your workstations might have already been infected. Your competitors have kindly shared with you, Indicators of Compromise (IoC) [3]. These are digital evidence that suggest that a system may have been breached. You have been tasked to improve the security infrastructure of your company. You are required to ensure that your systems have not been compromised and to create a secure environment, where highly confidential files can be stored securely. Finally, you would like to support other cybersecurity engineers and help them detect this specific attack.

Indicators of Compromise

The security experts that have studied this attack have identified the following characteristics:

- The virus makes use of a malicious library in order to encrypt itself and remain hidden. The attackers thought that it would be a good idea to implement the encryption code as a shared library in order to use it in their other malware applications. Experts have identified two different versions of this library. The first version has an MD5 [4] hash value of **`85578cd4404c6d586cd0ae1b36c98aca`** while the second version has SHA256 [5] hash value of **`d56d67f2c43411d966525b3250bfaa1a85db34bf371468df1b6a9882fee78849`**

- The ransomware that KozaliBear deployed asks for Bitcoin so that they will unlock the victim's files. The attackers use Bitcoin because they think it is anonymous. Victims should pay to the following wallet: **`bc1qa5wkgaew2dkv56kfvj49j0av5nml45x9ek9hz6`**

- Security experts have thoroughly investigated the attack and discovered that the attackers make use of an old virus that when attached to other programs contains the signature **98 1d 00 00 ec 33 ff ff fb 06 00 00 00 46 0e 10**.

- The ransomware that the attackers use, encrypts local files using the process:
    a. It first reads the entire content of a file: `example.txt`
    b. It then creates a new file based on the original filename and appends `.locked` to to the new filename: `example.txt.locked`
    c. It writes the encrypted content to the new file
    d. It deletes the original file.

- Other cybersecurity engineers have observed an increased amount of network traffic when their systems were infected by KozaliBear. By reverse-engineering this traffic from various infected workstations they found that the ransomware tries to download malware from popular online malware distribution platforms to further contaminate the victim's workstation.

1. Scanning for Infected Files

Your first task is to create an application that scans a workstation and searches for infected files. To discover infected files you will utilize the Indicators of Compromise of the previous section. The application will only act as a notification system. That is, it will not delete or quarantine any discovered infected files. It will only notify administrators that infected files exist. For this task you need to use the C programming language. Your application will receive as input a directory that it will then scan for infected files.

Implementation Details

For this task you are only allowed to use the standard library of C, as well as the OpenSSL library to compute hash values of files. More information about the OpenSSL library can be found in its man pages [6]. It is important to notice that your implementation should perform a recursive walk starting from the given directory and search for infected files in all subdirectories of the directory tree. For example, if the user wants to scan `/secret/data` then your system should find all infected files in the given directory (e.g. `/secret/data/malicious.out`) as well as in any subdirectories (e.g. `/secret/data/super/duper/secret/evil.dat`).

For each file you discover in the directory tree you need to compute its MD5 and SHA256 hash values and check if they match any of the hashes of the known malicious libraries. Additionally, you need to read the content of each file and search for either the signature of the known virus or the reported Bitcoin address. Any file found to contain these two values can be considered infected. When you have successfully processed all files, your application should provide a small report with information about the files it processed, as well as any potential discovered infected files. For example:

```
$ antivirus scan /home/ceo/Downloads

[INFO] [9046] [14-Mar-24 13:53:43] Application Started
[INFO] [9046] [14-Mar-24 13:53:43] Scanning directory
/home/ceo/Downloads
[INFO] [9046] [14-Mar-24 13:53:45] Found 3125 files
[INFO] [9046] [14-Mar-24 13:53:45] Searching…
[INFO] [9046] [14-Mar-24 13:53:55] Operation finished
[INFO] [9046] [14-Mar-24 13:53:55] Processed 3125 files. Found 7
infected

/home/ceo/Downloads/boot/grub/x86_64-efi/reiserfs.mod:REPORTED_VIRUS
/home/ceo/Downloads/malicious.so:REPORTED_MD5_HASH
/home/ceo/Downloads/snap/bare/current/dev/display.py:REPORTED_BITCOIN
/home/ceo/Downloads/snap/bare/current/dev/get.py:REPORTED_BITCOIN
/home/ceo/Downloads/snap/bare/current/dev/main.py:REPORTED_BITCOIN
/home/ceo/Downloads/kozi/v2/mal.so:REPORTED_SHA256_HASH
/home/ceo/Downloads/opt/google/chrome/libEGL.so:REPORTED_VIRUS
```

2. Detecting Potential Harmful Network Traffic

Your second task is to implement an application that scans all files inside a directory and attempts to discover if these files will potentially generate harmful network traffic. Your system will work on the domain level and will notify administrators that a file is potentially harmful if it attempts to interact with a malicious domain. Additionally, your system will be proactive. It will try to detect such files before they are executed and before any connection has been established to malicious domains. Again, you should not delete or quarantine any discovered files. You are only allowed to use the C programming language. Your application will receive as input a directory that it will then scan.

Implementation Details

Your application should find all files in a directory tree, extract all domains found in plaintext inside these files and test if they are malicious or not. As before, you should perform a recursive walk starting from the given directory and search for files in all subdirectories. In order to extract domains from the content of a file you should use regular expressions. You are free to use any regular expression you think is appropriate but you should be in position to explain it.

To classify domains into malicious or benign you should utilize Cloudflare's Malware and Adult Content Filter [7]. For each domain you discover you are expected to send requests to Cloudflare's endpoints to discover if the domain is malicious or not. To achieve this, you are required to make use of the highly popular libcurl library. You can find more information about the popular URL library in its documentation page [8].

When you have successfully processed all files, your application should provide a small report with information about the domains it detected, the files that contained each domain, as well as a decision if the domain is malicious or not. Your application should process all files it discovers regardless if they are binary or text files or if they are executable or not. However, in your final report you can optionally specify which files were executable.

```
$ antivirus inspect /home/ceo/

[INFO] [9046] [14-Mar-24 13:53:43] Application Started
[INFO] [9046] [14-Mar-24 13:53:43] Scanning directory /home/ceo/
[INFO] [9046] [14-Mar-24 13:53:45] Found 18312 files
[INFO] [9046] [14-Mar-24 13:53:45] Searching…
[INFO] [9046] [14-Mar-24 13:53:55] Operation finished
[INFO] [9046] [14-Mar-24 13:53:55] Processed 18312 files.

| FILE    | PATH                  | DOMAIN        | EXECUTABLE | RESULT  |
=============================================================================
| foo.exe | /home/ceo/docs/secret | www.google.com    | True  | Safe    |
| bar.txt | /home/ceo/hy457grade/ | alphaxiom.com     | False | Malware |
| libd.so | /home/ceo/Desktop/    | https://bbc.com   | False | Safe    |
| wget.sh | /home/ceo/aws/plugin  | biawwer.com       | True  | Malware |
```

3. Securing Valuable Files

Your third task is to create a secure enclave where stakeholders can place their most valuable data. This will be implemented in the form of a secure directory which will be continuously monitored and protected against ransomware. Once again, you are only allowed to use the C programming language. Your application will receive as input a directory that it will monitor until it is killed.

Implementation Details

Your application will create a secure enclave by monitoring file system events in the directory the user specified. You should monitor and print all these events in real time. Whenever a new filesystem event takes place, you should evaluate whether this event (along with previous events) indicates the presence of a ransomware that is trying to attack the secure enclave. When you detect such a behavior you are only required to print a message as a notification to the administrators. There is no need to carry out any other action.

In order to monitor for file system events, you are required to utilize the inotify API [9] and focus on the directories functionality. You should not monitor specific files since we don't know beforehand what files the board will place inside the enclave. In this task, you are not required to handle any subdirectories. You can assume that all files are placed directly into the main directory the user specified when it launched your application.

```
$ antivirus monitor /root/vault/

[INFO] [9046] [14-Mar-24 13:53:43] Application Started
[INFO] [9046] [14-Mar-24 13:53:43] Monitoring directory /root/vault/
[INFO] [9046] [14-Mar-24 13:53:43] Waiting for events...
File 'info.txt' was created
File 'info.txt' was opened
File 'info.txt' that was not opened for writing was closed
File 'passwords.txt' was opened
File 'passwords.txt' was accessed
File '.tmpSjxiska.dat' was deleted from watched directory
File 'passwords.txt.locked' was created
File 'passwords.txt.locked' was modified
File 'passwords.txt.locked' that was opened for writing was closed
File 'passwords.txt' was deleted from watched directory
[WARN] Ransomware attack detected on file passwords.txt
File '.tmpSIfwiunew.dat' was created
File 'studentGrades.csv' was opened
```

4. Protecting from Unauthorized Access

Your final task is to ensure that all documents that have been placed in the secure enclave of the previous step, do not fall into the wrong hands. There might be a chance that one of the workstations is infected or that one of the stakeholders turns rogue and tries to steal one of these files. The board would like to ensure that all files in the directory are encrypted and that no single individual can access the plaintext files on its own. Your supervisor tasked you with implementing a solution using Shamir's secret sharing scheme [10]. The encryption part of this design will be implemented by a different co-worker. There is no need for you to worry about it.

Implementation Details

Secret sharing works by splitting private information into smaller pieces - or shares - and then distributing those shares amongst a group or network. Each individual share is useless on its own but when all the shares are together, they reconstruct an original secret. The original secret in our case is the key that decrypts the files. Requiring all shares to reconstruct the original secret every time we want to access a file, seems impractical and inefficient. Instead, a threshold of minimum shares must be set to avoid unpredicted shareholder behavior.

For this task you are required to implement a secret sharing mechanism that would decrypt the files when at least three members of the company's board are present. There are ten members in the stakeholders' board. Only if any three (or more) of them are present, a file can be accessed. Otherwise, the secure enclave remains sealed. To share the password among the members of the board you have to implement a secret sharing method that relies on polynomial interpolation. More specifically you will write a C program that:

1. Constructs a 2nd degree polynomial $f(x) = a_2 \cdot x^2 + a_1 \cdot x + a_0$ where a_0 is the password and a_1, a_2 are randomly generated numbers. Note that if the secret has to be reconstructed by k entities, the polynomial degree must be k-1.

2. Gives each member of the board a tuple in the form of $(x_n, f(x_n))$. The first member of the board would take f(1) the second f(2), the third f(3), and the last one would take f(10). Note that f(0) results in $f(x) = a_2 \cdot 0 + a \cdot 0 + a_0 \Leftrightarrow f(0) = a_0$. Hence, f(0) is the secret password that is split into pieces and must not be shared.

3. Is able to reconstruct the original encryption key if at least 3 shares are provided as input. Note that these 3 shares can be any of the original 10 and not necessarily consecutive board members.

4. Provides two operation modes. One that splits the encryption key and generates the 10 shares, and one that reconstructs the key given at least 3 shares.

```
$ antivirus slice 156

[INFO] [9046] [14-Mar-24 13:53:43] Application Started
[INFO] [9046] [14-Mar-24 13:53:43] Generating shares for key ‘156’

(1, 313)
(2, 760)
(3, 1497)
(4, 2524)
(5, 3841)
(6, 5448)
(7, 7345)
(8, 9532)
(9, 12009)
(10, 14776)
```

```
$ antivirus unlock (1, 313) (4, 2524) (9, 12009)

[INFO] [9046] [14-Mar-24 13:53:43] Application Started
[INFO] [9046] [14-Mar-24 13:53:43] Received 3 different shares
[INFO] [9046] [14-Mar-24 13:53:44] Computed that a=145 and b=12
[INFO] [9046] [14-Mar-24 13:53:44] Encryption key is: 156
```

5. Disseminating Findings (BONUS + 1)

It is important to make your findings publicly available to help other cybersecurity engineers and lower the barrier for other researchers to understand the specific attack. This will make it easier to detect and stop this attack as early as possible, and even create defenses against it. One efficient way to identify and classify malware is the use of YARA rules [11]. A YARA rule is a description of a malware based on their patterns. Write a YARA rule to describe the KozaliBear attack described in the Introduction. Additionally, use the Arya tool [12] to generate pseudo-malicious files that match your YARA rule to test the implementation of your assignment. You can submit the YARA rule along with the command you used for the Arya tool in the README file of your assignment.

Notes

1. Your implementation should produce a single executable file that will provide various functionalities. Each part of this assignment should be implemented as a module of the same application. The user of your application will specify which module to execute using the appropriate verb (e.g. scan, secure, unlock) as a CLI argument.

2. This is not a group assignment. Each student should submit their own implementation and you are not allowed to work with each other. If you decide to use hosting services or version control systems (e.g. Git), do not forget to mark your repository as private.

3. Implement all the tasks of this assignment using the C programming language. You are free to develop and test your implementation on your personal device, however, the final version should work on CSD workstations.

4. A directory with various files has been created for you to test your application. You can download the test files from the course's website [13]. Please note that these tests are just indicative and that you should create your own test files to ensure your implementation is correct.

5. You can use the course's mailing list for any questions related to this assignment. Please provide a clear subject when sending an email.

6. Do not send any private messages to the teaching assistants. Other students may have the same question.

7. Do not send code snippets or files of your implementation to the mailing list. If you do so, your assignment will not be accepted and you will not be graded for the assignment.

8. For this assignment you need to provide a Makefile. The makefile should contain at least three rules. One that builds your system, one that runs it using your own test files and one that deletes build files (e.g. object files). Please clean the directories before submitting your assignment.

9. You should provide your own tests to demonstrate that your implementation is correct. This applies to all tasks and might involve creating simple executables.

10. You should provide a short README file that describes what parts of the assignment you have implemented, what you implemented differently and anything else that you consider important. Please keep this file relatively short.

11. Follow the steps described above and implement the assignment incrementally. This will be especially helpful for you. You can develop small building blocks and then integrate them into the final application. This will also help you with identifying and solving bugs.

12. Note that the submitted code will be tested for plagiarism using appropriate software.

13. You can submit the assignment by executing the command `turnin assignment_2@hy457 directory_name` where directory_name is the directory that contains the source code.

References

[1] https://en.wikipedia.org/wiki/Ransomware
[2] https://en.wikipedia.org/wiki/Cryptocurrency
[3] https://en.wikipedia.org/wiki/Indicator_of_compromise
[4] https://en.wikipedia.org/wiki/MD5
[5] https://en.wikipedia.org/wiki/SHA-2
[6] https://www.openssl.org/docs/manmaster/
[7] https://blog.cloudflare.com/introducing-1-1-1-1-for-families
[8] https://curl.se/libcurl/
[9] https://man7.org/linux/man-pages/man7/inotify.7.html
[10] https://en.wikipedia.org/wiki/Shamir%27s_secret_sharing
[11] https://yara.readthedocs.io/en/stable/index.html
[12] https://githu b.com/claroty/arya
[13] https://csd.uoc.gr/~hy457/resources/assignments/hy457_assignment_2_test_filesystem.zip