Secured System Engineering: Second project

April 16, 2025

Introduction

This assignment is to be completed in groups of 2. Note that for organisational reasons you must be in the same groups as you were for first assignment.

Due date: **2025-05-08 18h00**

1 Password cracking

1.1 Part I - Hash cracking

In this first part your goal is to crack each of the following password hashes. This may be useful : rockyou.txt[1].

```
MD5
Hash : f806fc5a2a0d5ba2471600758452799c

SHA-256
Hash : e83176eaefcc1ae8c4a23dbc73ebcf122f26cfb9ba5c7cf4763e96c1c38a6c6c

SHA-256 with salt
Hash : ac62977c69b2914679684489f417dbcc5bd923c858f8c068ac33bee86cc9a425
Salt : cf031883d7f9b72d

PBKDF2-HMAC-SHA256
Hash : 26db87b3a84b53cf4a7b0f10f5ca973586004b2e624db7336270f6ec4a773ee1
Salt : 25642ed7ebfc88f8c091cc4fda67c2e2
Iterations : 1000

Argon2
Hash : $argon2id$v=19$m=65536,t=3,p=4$kqVx8ohXApe2s3RlLlHGEw$4zkjrANAlrU1KjsJIYR6C+
    FJa3USw58Ufr81d5pdnhU
```

1.2 Part II - Bad password generator

You received a leak from a login database, for which passwords seem to have been generated by a password manager taking inspiration from Apple[2], but failing to properly compute the guessing entropy. Can you crack the hash of an unknown password (hash.txt for your group in the zip file)? What is the guessing entropy of the password generated by that password manager? What are the weaknesses in the storage of the password hashes?

[1] Slower encryption algorithms will have the solution somewhat close to the beginning of this wordlist.

[2] `https://rmondello.com/2024/10/07/apple-passwords-generated-strong-password-format/`

1.3 Part III - Auth cracking

You managed to find the authentication program used to access a server. Your goal is to find the password for the provided application "auth_challenge" in order to access the server. It is a 64bit linux application so it won't run on the VM.

The goal is to find this password. Don't patch the program to bypass the password check. The goal is to be able to access the (hypothetical) server.

1.4 Deliverable

For each part give the requested answers. Describe your approach and methodology. Mention the tools and commands you used. If you created a script, provide it. If you've used a heavy wordlist/table/... **don't** provide it. Just mention its name or/and where you've found it.

2 YouGonnaCry - Malware Analysis

You are responsible for ensuring the malware stays contained and does not cause damage. *Do not attempt* to make this malware functional outside of the educational setting. Any misuse or negligent handling of this tool may result in disciplinary or legal consequences.

You are given a Linux ELF 64 bits binary of `(You)GonnaCry`, an educational ransomware designed to simulate common real-world techniques found in modern malware campaigns. Your mission is to reverse engineer, analyze, and document the ransomware's behavior via:

- Static Analysis
- Dynamic Analysis
- MITRE ATT&CK Technique Mapping[3]

You must perform both static and dynamic analysis and document the findings. While you are free to use the tools and methods of your choice, your report must clearly explain the reasoning behind your analysis and illustrate the ransomware's behavior through examples, screenshots, or logs.

You are required to map **at least four (4)** MITRE ATT&CK techniques based on your findings, with **at least one (1)** based on observed dynamic behavior (80% of the grade if good explanation). The rest can be inferred statically or dynamically.

2.1 Requirements

- `GLIBC_2.33` and `GLIBC_2.34` (Tested on Ubuntu 24)
- Install `sudo apt-get install libssl-dev` - OpenSSL 3.0.13 30 Jan 2024 (Library: OpenSSL 3.0.13 30 Jan 2024

2.2 Part I - Static Analysis

Perform this part *without executing* the binary. Focus on understanding its structure, intentions, and logic.

Recommended tasks:

1. Use tools like 'strings', 'readelf', 'objdump', 'radare2', or 'Ghidra' (last one recommended)

[3] `https://attack.mitre.org/`

2. Compilation informations (e.g: dynamic vs static, striped, PIE, etc)
3. Decompile or disassemble to inspect:
    - File targeting logic
    - Encryption algorithm and key management
    - Evidence of ransom note creation
    - Hardcoded paths, extensions, or credentials
    - ...
4. Map static indicators to MITRE ATT&CK techniques, e.g.:
    - T1486 – Data Encrypted for Impact
    - T1083 – File and Directory Discovery
    - ...

2.3 Part II - Dynamic Analysis

Run the binary **only inside an isolated sandbox or VM**. Use system monitoring tools to observe its behavior in a controlled environment.

Recommended sandboxing: ‘chroot‘, Docker (with read-only volumes), LXC, QEMU/KVM VM.

You *can* also do some monitoring with:

- ‘strace‘, ‘ltrace‘, ‘auditd‘, ‘fanotify‘, ‘inotify‘ (for file operations, if present)
- ‘tcpdump‘, ‘ss‘, ‘iptables‘ (for network behaviors, if present)
- LiMe dumps with Volatility.

Recommended tasks:

1. Observe:
    - Which files get encrypted
    - Location of dropped ransom notes
    - Attempts at persistence (e.g. ‘.bashrc‘, cron jobs)
    - etc.
2. Detect behaviors matching MITRE ATT&CK, such as:
    - T1070.004 – Indicator Removal on Host
    - T1037 – Boot or Logon Initialization Scripts
    - ...

Note: The malware might implement some anti-debugging techniques (*check the links* ☹).

Note: You might need to setup the environment to reveal all behaviors. (see `setup_files.py`)

2.4 Deliverable

Prepare and submit a report including:

1. Executive Summary: What does this malware do?
2. MITRE Mapping: List of techniques used with justifications.
   For each MITRE technique found, you must explain how it is implemented and how you found them.
   (a) Static Analysis Findings: Behavior and key artifacts
   (b) Dynamic Analysis Observations: Runtime behavior, logs, etc.
3. Defensive Insights: How could this be detected, prevented, or contained?

Bonus - Offensive Insights: How could you improve the malware ?

2.5 Useful links

2.5.1 Isolation

- https://github.com/capeprivacy/capejail/
- https://ubuntu.com/blog/kvm-hyphervisor
- https://www.qemu.org/
- https://virt-manager.org/
- https://ubuntu.com/download/desktop
- https://jon-lein.github.io/Docker-Malware-Tools/

2.5.2 Binary Analysis

- https://ghidra-sre.org/
- https://htmlpreview.github.io/?https://github.com/NationalSecurityAgency/ghidra/blob/stable/GhidraDocs/CheatSheet.html
- https://www.tripwire.com/state-of-security/ghidra-101-binary-patching
- https://github.com/rshipp/awesome-malware-analysis

2.5.3 Obfuscation/AntiDebug

- https://reverseengineering.stackexchange.com/questions/3323/how-to-prevent-upx-d-on-an-upx-packed-executable
- https://github.com/upx/upx/tree/v4.0.1
- https://reverseengineering.stackexchange.com/questions/3335/decoding-the-upx-elf-header-file
- https://gchq.github.io/CyberChef/
- https://www.javatpoint.com/block-cipher-vs-stream-cipher
- https://en.wikipedia.org/wiki/Binary-to-text_encoding

- https://attack.mitre.org/
- https://medium.com/@andreabocchetti88/detect-vm-environment-using-cpuid-f461e89f81c0
- https://seblau.github.io/posts/linux-anti-debugging
- https://rayanfam.com/topics/defeating-malware-anti-vm-techniques-cpuid-based-instructions/

3 BREACH attack

For this exercise, you will take advantage of the BREACH attack[4].

3.1 How to run the exercise

You do not need the VM but `Docker` and `python3`. To build the docker, run the command `docker build . -t breach` inside the directory BREACH. To run it, use `docker run -d -p 5000:5000 -p 5001:5001 breach`. Do not forget to kill your container when you have finished using it with `docker kill <container_name>`.

3.2 Explanation of the setup

At the address `https://localhost:5000` runs a server with the following pages:

1. At `/` you have a home page
2. At `/singin` you can connect to an existing account
3. At `/singup` you can create an account
4. At `/logout` you can logout from your session
5. At `/secret?username=<username>` you can have a link to retrieve the flag
6. At `/get_flag?csrf_token=<csrf_token>` you have the flag if you have the CSRF token of the oracle. This page is only to check if you found the right CSRF token. Normally, you need to be connected as the oracle to have access to the content. The CSRF token is not normally used as a connection token.

At the address `http://localhost:5001` runs an oracle. This oracle simulates a client over which you have some amount of control (through a technical or social engineering attack). It is connected to the server with its credentials and can make requests to it. On the home page (`/`), you have a form where you can submit a request string. The oracle will make the request to the server. For example, if you request `get_flag`, the oracle will request `https://localhost:5000/get_flag`. Then, the oracle will respond with the content length of the response received from the server, simulating what a network adversary could observe. The file `breach_attack.py` shows an example of request to the oracle and prints the response.

3.3 Task

Your goal is to find the secret CSRF token of the oracle and request the flag to the server with it. The CSRF token will always have the same size and only contains lowercase hexadecimal characters. A simple brute force over the CSRF token without using the oracle will not be accepted as a solution. To help you get started, here are a few questions to answer and include in your report. (They can be answered with one or two sentences.)

1. What is the vulnerability used in BREACH ?

[4] `https://en.wikipedia.org/wiki/BREACH`

2. How can you find (without looking at the code of the server) the length of the CSRF token assuming they have a fixed length ?
3. Which page of the server can be used to take advantage of the vulnerability and why ?
4. How the content length provided by the oracle can be useful ?
5. Which requests will you make to the oracle to retrieve the CSRF token ?
6. The oracle returns the HTTP `Content-Length`. How could you observe this as an adversary?
7. (To be answered after making the attack) Which TLS feature could prevent your attack? (BONUS: How could you modify your attack to bypass this mitigation?)
8. (BONUS - May be answered after making the attack) This exercise stops after recovering the CSRF token. In an actual attack, how could you exploit knowledge of the token?
9. (BONUS - May be answered after making the attack) An element in the server is misused. How can you modify the server so that it no longer has this vulnerability but still supports compression?

3.4 Deliverable

Complete the file `breach_attack.py` with your attack. In the end, it should print the flag (one or a few requests with the CSRF tokens guessed to the server). Do not hesitate to print a progression of your program. The attack should take less than 5 minutes on a regular computer. (It is possible to do it in less than 1 minute). We ask you to use only standard Python libraries[5] (except for the `requests` library already imported). We are not going to install a library with `pip` to test your code! We are going to test your code with different CSRF tokens (but with the same size) and credentials (you are not supposed to log in as an oracle or find its password). Add to the report the answers to the different questions asked previously.

Deliverable

This project **must** be done **in groups of 2 students**. **Your group should be the same as the one declared for the first assignment**. The group is tasked to provide a zip file containing:

- A PDF report explaining your solutions to the 3 different questions.
- When applicable: the code or script of your solution.
- Any other material that you deem relevant to understand your solution.

The deadline for this project is on the 8th May 2025 at 18h00. You will be able to submit your solutions for the project on the Moodle page of the course.

[5] `https://docs.python.org/3/library/index.html`