PIPETO

Customer:
The Stone Corporation
2025-05-22
v0.0

Contact:
Enzo IFFIS + Savinien PETITJEAN

enzo.iffis@epitech.eu

Table of Contents

Introduction 2

Black-Box Audit 3

White-Box Audit 4

Fix 5

Vulnerability Overview 6

Vulnerability Details 9

- Admin Gate Bypass trigger_emergency_shutdown (High) 9
- Config File Buffer Overflow load_config (High) 10
- Hardcoded Admin Password activate_emergency_protocols (High) 11
- Stack Overflow monitor_radiation_levels (Medium) 12
- Bad Encryption check_reactor_status (Medium) 13
- Bad Encryption send_status_report (Medium) 14
- Bad comparison run_turbine (lib) (Medium) 15
- Int overflow turbine_temperature (Medium) 16
- Log Injection & Secret Leak log_system_event (Medium) 17
- Use After Free check_cooling_pressure and bad string compare (Medium) 18
- Format String Attack load_pipetorc (Low) 19

Conclusion 20

Introduction

This report contains all the information regarding our PIPETO project from the G-SEC-200 module. During this project, the PIPETO binary was provided to us, and using various tools, we were able to practice reverse engineering and fuzzing in order to find vulnerabilities and, in the process, flags (strings of characters enclosed between two curly braces { }).

This report will therefore cover the Black-Box phase carried out during the first two weeks, followed by the White-Box phase over the next two weeks, and finally, the Fix phase during the last two weeks.

Here is also a list of the tools we were able to use throughout the duration of this project:

- Ghidra (A disassembler and decompiler used to analyze binary files such as PIPETO)
- Cutter (Also a decompiler, but with a graphical view and an integrated debugger)
- GDB (A compiler and debugger)
- Clang (A compiler that provides additional insights into potential security vulnerabilities)
- AFL (A brute-force fuzzer)
- Strings (A linux command which can extract readable strings from a binary file)
- Fanalyzer (Used to detect various warnings and bugs in the code)

Black-Box Audit

When the PIPETO binary is launched, a command-line interpreter starts in the following form : pipeto > and when the help command is entered, a list of commands that can be executed by the binary is displayed.

Here is the list of commands that contain vulnerabilities :

[activate_emergency_protocols => {Emergency protocols activated, you are now admin !}, Hardcoded Password]

[load_fuel_rods => {The secret stone is here !}, Hardcoded Value]

[run_diagnostic => {SECRET DIAGNOSTIC KEY}, Hardcoded Value]

[turbine_temperature => {ERR0R TURBINE WILL EXPLODE}, Int Overflow]

[turbine_remote_access => {ACCESS_GRANTED}, Race condition]

[set_reactor_power => {12EXPLOSION34}, Int Overflow]

[run_turbine => {ERR0R TURB1NE CAN'T ST0P}, Int Underflow]

[simulate_meltdown => {MELTDOWN1234}, Vulnerabilty with rand function]

[.pipetorc file => File who can execute shell commands so he can launch a shell]

[monitoring_radiation_levels => flag not found during this phase but it's a buffer overflow]

[configure_cooling_system => no flag in this command, String attack, Command Injection]

[init_reactor => Important data encrypted with ROT13, weak encryption]

[send_status_report => Important data encrypted with base64, weak encryption]

[log_system_event => flag not found, Log poisoning]

These vulnerabilities were discovered through manual testing and with the help of Ghidra, which allowed us to identify hardcoded values as well as buffer overflows and underflows. During this Black-Box phase, we managed to find a total of 7 flags as well as 4 vulnerabilities.

White-Box Audit

After the Black-Box phase, we entered the White-Box phase, meaning that the source code of the binary was provided to us. This allowed us to understand precisely how the binary worked and to observe how values were being handled. Moreover, it enabled us to identify the most obvious vulnerabilities, which was especially helpful for the Fix phase that followed the White-Box analysis.

The function activate_emergency_protocols() prompted us for a password: admin123, which we found by running strings on the binary during the Black-Box phase.

If the correct password is entered, it sets the boolean IS_ADMIN to True and also returns the flag: {Emergency protocols activated, you are now admin !}

This vulnerability is known as a Global Variable issue. The boolean is returned to a structure that is used in several commands.

Here are the vulnerabilities caused by this issue :

[unlock_secret_mode => {ADMIN4242}, Global Variable]

[trigger_emergency_shutdown => {SHUTDOWN}, Global Variable]

We also found several other vulnerabilities afterwards.

[load_config => {Correct password! Welcome, admin.}, Config Injection & Buffer Overflow]

[log_system_event => {SECRET_LOG_12PIERRE34}, Log Injection]

[monitor_radiation_levels => {The stone isn't in the pocket anymore ...}, Buffer Overflow]

[check_cooling_pressure => {Sensitive Data}, Use after Free]

[load_pipetorc => no flag but printf not secured, Format String Attack]

During this White-Box phase, with the help of the binary’s source code, we were able to find 6 additional flags.

Fix

Each vulnerability identified during the source code audit is directly followed by a proposed remediation strategy. This structure ensures a clear understanding of not only the nature and risk of each issue, but also how it has been or should be addressed in context.

The fixes include code hardening techniques such as bounds checking, removal of hardcoded secrets, format string protections, proper use of cryptographic primitives (e.g., OpenSSL), and enforcing stricter access control logic.

By integrating these fixes directly alongside their corresponding issues, we ensure traceability between the original weakness and the corrective action taken, thus improving auditability and accountability.

Vulnerability Overview

In the course of this penetration test **3 High**, **7 Medium** and **1 Low** vulnerabilities were identified:

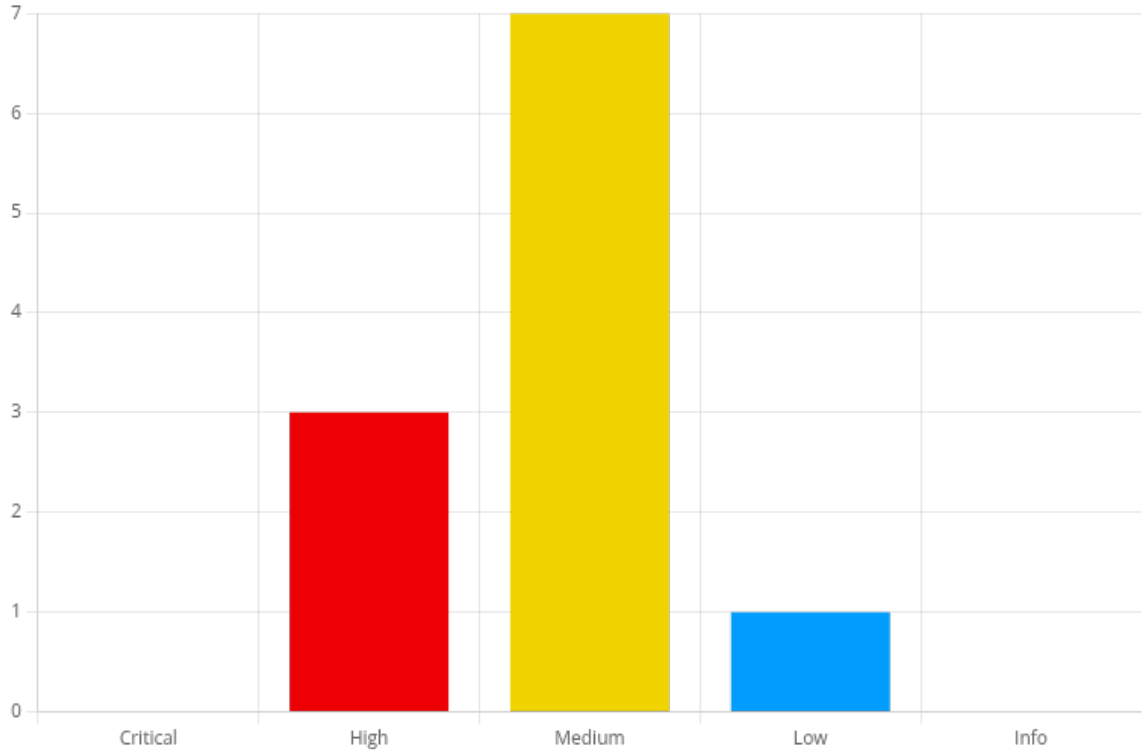

Figure 1 - Distribution of identified vulnerabilities

A tabular overview of all vulnerabilities identified:

Vulnerability	**Criticality**
Admin Gate Bypass trigger_emergency_shutdown	**High**
Config File Buffer Overflow load_config	**High**
Hardcoded Admin Password activate_emergency_protocols	**High**
Stack Overflow monitor_radiation_levels	**Medium**
Bad Encryption check_reactor_status	**Medium**
Bad Encryption send_status_report	**Medium**
Bad comparison run_turbine (lib)	**Medium**
Int overflow turbine_temperature	**Medium**
Log Injection & Secret Leak log_system_event	**Medium**
Use After Free check_cooling_pressure and bad string compare	**Medium**
Format String Attack load_pipetorc	**Low**

A list of all vulnerabilities including a brief description:

1. Admin Gate Bypass trigger_emergency_shutdown (High: 7.3)

Function: trigger_emergency_shutdown

Impact: Grants flag {SHUTDOWN}

2. Config File Buffer Overflow load_config (High: 7.3)

Function: load_config

Impact: Malicious config.ini causes a crash → Flag {Correct password! Welcome, admin.}

3. Hardcoded Admin Password activate_emergency_protocols (High: 7.3)

Function: activate_emergency_protocols

Impact: Grants access to unlock_secret_mode → Flag {ADMIN4242}

4. Stack Overflow monitor_radiation_levels (Medium: 5.9)

Function: monitor_radiation_levels

Impact: Debugging reveals secret_function() → Flag {The stone isn't in the pocket anymore ...}

5. Bad Encryption check_reactor_status (Medium: 5.9)

Function: check_reactor_status

Impact: Grants secret

6. Bad Encryption send_status_report (Medium: 5.9)

Function: send_status_report

Impact: Grants secret

7. Bad comparison run_turbine (lib) (Medium: 5.9)

Function: run_turbine

Impact: Flag {ERR0R TURB1NE CAN'T ST0P} shown unconditionally

8. Int overflow turbine_temperature (Medium: 5.9)

Function: turbine_temperature

Impact: Flag {ERR0R TURBINE WILL EXPLODE} shown unconditionally

9. Log Injection & Secret Leak log_system_event (Medium: 5.9)

Function: log_system_event

Impact: Flag {SECRET_LOG_12PIERRE34} is exposed in system.log

10. Use After Free check_cooling_pressure and bad string compare (Medium: 5.9)

Function: check_cooling_pressure

Impact: Flag {Sensitive Data} shown unconditionally

11. Format String Attack load_pipetorc (Low: 3.8)

Function: load_pipetorc

Impact: Format string attack → memory disclosure or RCE

Vulnerability Details

1. Admin Gate Bypass trigger_emergency_shutdown

Remediation Status:
Criticality: High
CVSS-Score: 7.3

Overview

Function: trigger_emergency_shutdown

Impact: Grants flag {SHUTDOWN}

Description

Issue: Admin access is check with IS_ADMIN global variable that is editable in run time

Recommendation

Fix:

- Replace global admin flag with secure session validation
- Centralize privilege checks

2. Config File Buffer Overflow load_config

Remediation Status:
Criticality: High
CVSS-Score: 7.3

Overview

Function: load_config

Impact: Malicious config.ini causes a crash → Flag {Correct password! Welcome, admin.}

Description

Issue: No input size check when reading config.ini

Recommendation

Fix:

- Use fgets with proper bounds
- Sanitize input length and format

3. Hardcoded Admin Password activate_emergency_protocols

Remediation Status:
Criticality: High
CVSS-Score: 7.3

Overview

Function: activate_emergency_protocols

Impact: Grants access to unlock_secret_mode → Flag {ADMIN4242}

Description

Issue: Password "admin123" is hardcoded and easily retrievable via strings.

Recommendation

Fix:

- Hash hardcoded password
- Replace global IS_ADMIN with a struct passed to commands

4. Stack Overflow monitor_radiation_levels

Remediation Status:
Criticality: Medium
CVSS-Score: 5.9

Overview

Function: monitor_radiation_levels

Impact: Debugging reveals secret_function() → Flag {The stone isn't in the pocket anymore ...}

Description

Issue: Unbounded input leads to segmentation fault

Recommendation

Fix:

- Bound input sizes

5. Bad Encryption check_reactor_status

Remediation Status:
Criticality: Medium
CVSS-Score: 5.9

Overview

Function: check_reactor_status

Impact: Grants secret

Description

Issue: Bad custom encrypting function used

Recommendation

Fix

- Use openssl library to encrypt message with

6. Bad Encryption send_status_report

Remediation Status:
Criticality: Medium
CVSS-Score: 5.9

Overview

Function: send_status_report

Impact: Grants secret

Description

Issue: Bad base64 encrypting function used

Recommendation

Fix

- Use openssl library to encrypt message in sha256

7. Bad comparison run_turbine (lib)

Remediation Status:
Criticality: Medium
CVSS-Score: 5.9

Overview

Function: run_turbine

Impact: Flag {ERR0R TURB1NE CAN'T ST0P} shown unconditionally

Description

Issue: Compare rotations with 0 and > 15

Recommendation

Fix:

- Print error if rotations is > 15

8. Int overflow turbine_temperature

Remediation Status:
Criticality: Medium
CVSS-Score: 5.9

Overview

Function: turbine_temperature

Impact: Flag {ERR0R TURBINE WILL EXPLODE} shown unconditionally

Description

Issue: Degrees is a int and is used with strtol that return a long long

Recommendation

Fix:

- Change degress type to long long

9. Log Injection & Secret Leak log_system_event

Remediation Status:
Criticality: Medium
CVSS-Score: 5.9

Overview

Function: log_system_event

Impact: Flag {SECRET_LOG_12PIERRE34} is exposed in system.log

Description

Issue: User can write the string "leak" in the log file that expose a flag

Recommendation

Fix:

- Lock file using flock()

10. Use After Free check_cooling_pressure and bad string compare

Remediation Status:
Criticality: Medium
CVSS-Score: 5.9

Overview

Function: check_cooling_pressure

Impact: Flag {Sensitive Data} shown unconditionally

Description

Issue: Always compares "Pressure OK" to itself → always true and is used after free

Recommendation

Fix:

- Compare strcmp with 0
- Free after the comparison

11. Format String Attack load_pipetorc

Remediation Status:
Criticality: Low
CVSS-Score: 3.8

Overview

Function: load_pipetorc

Impact: Format string attack → memory disclosure or RCE

Description

Issue: printf(user_input) directly called

Recommendation

Fix:

- Use printf("%s", input)

Conclusion

This security assessment of the Pipeto binary was conducted in two distinct phases: an initial Black Box analysis, followed by a comprehensive White Box source code review.

During the Black Box phase, several vulnerabilities were identified through behavioral analysis and binary inspection, allowing partial privilege escalation and unintended access to sensitive data. These findings justified a deeper investigation under White Box conditions.

The subsequent source code audit confirmed and expanded upon the initial discoveries. Multiple critical issues were identified, including hardcoded credentials, unsafe buffer handling (leading to potential overflows), inadequate logging protections, and insecure format string usage. Each of these vulnerabilities represented a tangible risk to system confidentiality, integrity, or availability.

To address these concerns, targeted remediations were implemented. These include:

- Replacing plaintext credentials with hashed password verification using cryptographic primitives (OpenSSL).
- Hardening file input/output operations to prevent injection and overflow.
- Sanitizing and locking sensitive resources such as log files.
- Mitigating memory corruption risks and improving code structure and data encapsulation.

Overall, the Pipeto application has undergone significant security improvements. While the current posture is notably more robust, we recommend the adoption of secure development lifecycle (SDL) principles moving forward, including static analysis, fuzz testing, and secure coding guidelines.

This assessment demonstrates the value of combining external behavioral testing with internal code visibility. Such a dual-phase approach ensures both the detection of surface-level exploits and the resolution of underlying architectural weaknesses, thereby reinforcing long-term software resilience.