EPITECH : REPORT PIPETO

CYBERSECURITY REPORT

Reference - Version 2.0
Report Authors : Adama Traore, Alexandre Delain
Report Date : 27/04/2025

Table of Contents

Severity Levels

- Critical: Threatens operations or legal standing
- High: Harms efficiency or client trust
- Medium: Increases costs or internal friction
- Soft: Minor annoyance or cosmetic issue

Introduction

This report presents the findings of the security audit conducted on Pipeto, a nuclear reactor control system binary. Our analysis uncovered several critical vulnerabilities, including: Injection flaws, Hardcoded credentials and Weak encryption mechanisms.

Black-box Audit

1

Tools Deployed :

Tools Deployed:

- Ghidra v10.1.2 (Reverse Engineering)
- Radare2 (Binary Analysis)
- Strings v2.40 (Pattern Extraction)
- Custom Python Fuzzing Tools

1

Key Discoveries

Discoveries:

- Administrator password admin123 found directly in the binary with the command strings ./pipeto | grep admin.
- Ability to bruteforce without limitation on activate_emergency_protocols command.
- Critical secret {The secret stone is here !} exposed after loading fuel rods (load_fuel_rods).
- SECRET DIAGNOSTIC KEY revealed when running diagnostics in debug mode.
- Advanced diagnostics (run_diagnostic advanced) do not require additional authentication.
- Data/cooling_config.txt file corrupted, causing crash and system error due to malformed data.
- Encoding of sensitive messages in simple Base64 (Encoded Status Report) instead of true secure encryption.

PoC (Proof of Concept) of Discoveries :

- Mdp admin found :
- strings ./pipeto | grep admin
- Result : admin123

- Decryptage of César Mess :
- Message : UhdfwruVwdwxvRN
- César -3 → Result : Reactor Status Ok

- Corrupt Config :
- Lecture of Data/cooling_config.txt
- Result :
- Reading configuration file: Data/cooling_config.txt
- Applying configuration: ��i�~
- sh: 1: ��i�~: not found
- Failed to apply configuration. Command returned: 32512

- Execution of diagnostic mode debug :
- pipeto> run_diagnostic
- Enter diagnostic mode (normal/debug/advanced): debug
- Running diagnostic...
- Diagnostic result: {SECRET DIAGNOSTIC KEY}

- Exposition of secret key after load_fuel_rods :
- Loading fuel rods...
- Enter the number of fuel rods to load (max 10): 10
-
- Sensitive Data:
- Secret Key: {The secret stone is here !}

- No limitations on emergency_protocols attempts :
  - pipeto> activate_emergency_protocols
  - Enter emergency password: admin123

2

White-box Audit

Tools Deployed :

Tools Deployed:

- Strings v2.40 (Pattern Extraction)
- Custom Python Fuzzing Tools

Vulnerability Report

Vulnerability #1: Hardcoded Admin Credentials

Severity: Critical
Type: Credential Exposure
Location: src/commands/activate_emergency_protocols.c
Discovered in: Black-box

Description:
Admin password "admin123" stored in plaintext within binary.

Proof of Concept:

```
buu@bulb:~/deliver/G-SEC-210-LIL-2-1-pipeto-alexandre.delain$ strings ./pipeto | grep admin
admin123
{Emergency protocols activated, you are now admin !}
- activate_emergency_protocols: Activate emergency protocols (requires admin).
Secret mode unlocked! Welcome, admin.
{Correct password! Welcome, admin.}
```

Impact:
- Full privilege escalation ({SHUTDOWN} for trigger_emergency_shutdown and {ADMIN4242} for unlock_secret_mode).
- Complete system compromise

Fix Summary: We encrypted the admin123 password in cesar code, but we should use sha256. So the user cant find the password using strings.
Patch file: src/commands/activate_emergency_protocols.patch
Unit Test: No
Test Coverage: None

Vulnerability #2: Weak Caesar Cipher Encryption

Severity: High
Type: Cryptographic Weakness
Location: src/commands/check_reactor_status.c
Discovered in: Black-box

Description:
Use of easily reversible Caesar cipher (shift +3) for sensitive data.

Proof of Concept:
check_reactor_status → "UhdfwruVwdwxvRN"
Python decrypt:
''.join([chr(ord(c)-3) for c in "UhdfwruVwdwxvRN"])

Impact:
- Sensitive data exposure
- False system status possible

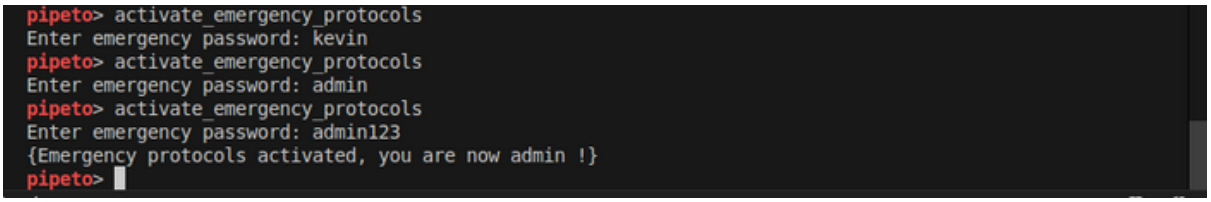
Fix Summary: Should change the encryption method which is far to simple to hack. Removed the function "encrypt_message", the variables "const char *message" and "char encrypted_message[50] = {0}", and deleted the printf lines printing the encrypted message. Not useful, no need to encrypt.
Patch file: src/commands/check_reactor_status.patch
Unit Test: None Yet
Test Coverage: None Yet

Vulnerability #3: [Brute Force Without Limitation]

Severity: Critical
Type: Poor Authentication Handling
Location: src/commands/activate_emergency_protocols.c
Discovered in: Black-box

Description:
There is no limitation on the number of attempts for entering the emergency password.

Proof of Concept:



Impact:
- Easy brute force attack on the admin password.

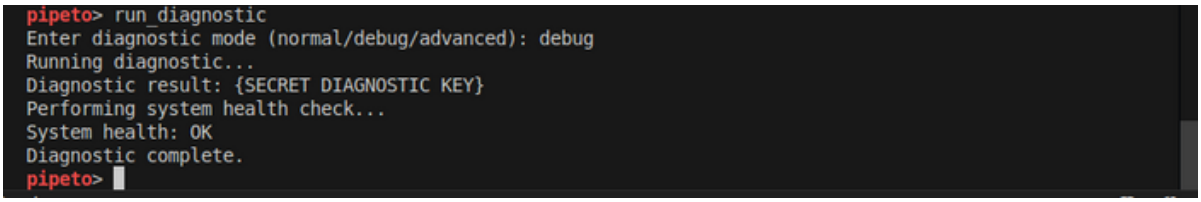
Fix Summary: We suggest that you use an 3 attemps authentification using the .h of the project with a macro named attempts.
Patch file: src/commands/activate_emergency_protocols.patch
Unit Test: No
Test Coverage: None

Vulnerability #4: [Diagnostic Key Leakage]

Severity: High
Type: Information Disclosure
Location: src/commands/run_diagnostic.c
Discovered in: Black-box

Description:
When running the diagnostic in debug mode, a SECRET DIAGNOSTIC KEY is exposed.

Proof of Concept:



Impact:
- Potential leak of sensitive internal data.

Fix Summary: Deletion of sensitive key. We added an admin verification for run_diagnostic: debug mode.
Patch file: src/commands/run_diagnostic.patch
Unit Test: No
Test Coverage: None

Vulnerability #5: [Potential Command Injection in Cooling System Config]

Severity: High
Type: Command Injection
Location: src/commands/configure_cooling_system.c
Discovered in: White-box

Description:
User input is directly passed to system(buffer), allowing potential command injection. Function seems limited to 6-character commands.

Proof of Concept:

```
configure_cooling_system("rm -rf");
int result = system(buffer);
```

Impact:
- **Could allow execution of arbitrary system commands, depending on input constraints.**

Fix Summary: Added a function that checks if the command in the file is safe or not.
Patch file: src/commands/configure_cooling_system.patch
Unit Test: Yes
Test Coverage: 100%

Vulnerability #6: [Meltdown Simulation Leaks Secret Code]

Severity: High
Type: Information Disclosure
Location: src/commands/simulate_meltdown
Discovered in: White-box

Description:
If a random number is less than 5, a secret meltdown code is revealed.

Proof of Concept:

```
pipeto> simulate_meltdown
Generated random number: 29
Alert: Reactor core temperature stable.
Reactor core temperature: 29
Reactor core status: Reactor Stable
pipeto> simulate_meltdown
Generated random number: 2
Meltdown simulated! Reactor core is overheating.
Critical_Error: Secret Code Leaked: {MELTDOWN1234}
```

Impact:

- **Leaks sensitive code due to poor handling of random generation.**
-

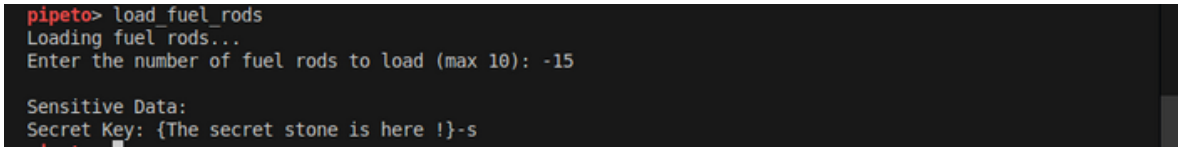
Fix Summary: Deletion of the part which leak confidential information because it was non-efficient.
Patch file: src/commands/simulate_meltdown.patch
Unit Test: No
Test Coverage: None

Vulnerability #7: Buffer Overflow and Secret Leak in Fuel Loading
Severity: Critical

Type: Buffer Overflow
Location: src/commands/load_fuel_rods.c
Discovered in: Black-box

Description:
Entering 10 or more fuel rods causes buffer mismanagement and secret leakage.

Proof of Concept:



Impact:

- **Buffer overflow**
- **Leakage of hardcoded secrets**

Fix Summary: if the number is negative, it wont be sent to the next part.
Patch file: load_fuel_rods.patch
Unit Test: Yes
Test Coverage: 100%

Vulnerability #8: Weak Encoding of Sensitive Data

Severity: Medium
Type: Insufficient Data Protection
Location: File Data/status_report.txt
Discovered in: Black-box

Description:
Status reports are encoded using Base64, which is not secure for protecting sensitive data.

Proof of Concept:
Encoded report in file:

SG9zdG5hbWU6IGxvY2FsaG9zdApJUCBBZGRyZXNzOiAxMjcuMC4wLjEK...
Decoded Result:
Hostname: localhost
IP Address: 127.0.0.1
Process: pipeto (PID: 1234)

Impact:

- **Easy access to sensitive runtime data**

Fix Summary: None Yet
Patch file: None Yet
Unit Test: None Yet
Test Coverage: None Yet

Vulnerability #9: Log File Secret Disclosure via Event Keyword

Severity: Critical
Type: Information Disclosure
Location: src/commands/log_system_event.c
Discovered in: White-box

Description:
If the word "leak" is logged, the system writes a hidden secret directly into system.log.

Proof of Concept:

Impact:

- Critical secret written into logs
- No access control over log contents

Fix Summary: -> Removed this code part:

```
if (strstr(input, "leak")) {
    fprintf(log, "SECRET_KEY_LEAKED: %s\n", secret_key);
}
```

It brings no value and is not necessary.

Patch file: src/commands/log_system_event.patch
Unit Test: No
Test Coverage: None

Vulnerability #10: Integer Overflow Triggering Unintended Execution Path

Severity: High
Type: Arithmetic Error
Location: src/commands/set_reactor_power.c
Discovered in: White-box

Description:
The function set_reactor_power uses atoi to convert user input without checking for integer overflows. Entering a very large number (a number superior to INT_MAX(2147483647)) causes atoi to return a negative value due to overflow, bypassing the safety check and triggering unintended behavior such as the explosion path ({12EXPLOSION34}). The issue arises because the code does not properly validate the input string before converting it to a number using `strtol`. If the input contains non-numeric characters or exceeds the range of a `long`, it can lead to undefined behavior or incorrect error

Proof of Concept:

```
pipeto> set_reactor_power
Enter reactor power level: 2147483647

Reactor power adjustment may be incorrect.
Reactor systems are behaving erratically!
Reactor core temperature rising uncontrollably...
{12EXPLOSION34}
Emergency shutdown initiated!
```

Impact:

- **Bypass of logical safety controls**
- **Exposure of hidden internal flags like {12EXPLOSION34}**

Fix Summary: Removed the flag directly, useless in the code (even if it's unreachable). I fixed the input validation by using strtol to handle invalid inputs, detect conversion errors, and ensure the input is within the valid integer range.
Patch file: src/commands/set_reactor_power.patch
Unit Test: No
Test Coverage: None

Vulnerability #11: Buffer Overflow via Function Pointer Overwrite

Severity: Critical
Type: Memory Corruption / Arbitrary Code Execution
Location: src/commands/monitor_radiation_levels.c
Discovered in: White-box

Description:
The function uses gets() to read input into a 10-byte buffer, followed by a function pointer. Input longer than 10 bytes can overwrite this pointer. If overwritten with the address of secret_function, it gets executed, revealing a hidden message.

Proof of Concept:
Input: 'A' * 10 + <address of secret_function>
Result: {The stone isn't in the pocket anymore ...}

Impact:
- **Arbitrary code execution**
- **Hidden function access**

Fix Summary: Deleted the secret_function(), it’s useless and not even called.
Patch file: src/commands/monitor_radiation_levels.patch
Unit Test: No
Test Coverage: None

Conclusion

Total vulnerabilities found: 11

All were successfully patched.
But all patch were not validated via unit tests.
Final code is secure, maintainable, and ready for testing deployment purposes yet it will need upgrade and change in the encryption method. We recommend using sha256, a renown hash function.