

Challenge SSTIC 2026 - Step 2 : a core lock

nebucca

Hello analyst,

As you may suspect, Aegis Tech have been compromised, for quite some time by hackers. It cannot be stressed enough how critical it is to regain control of this system. You have been cleared to access any leaked files while we continue to assess the damages.

First thing first, you can find a description of SAFE here. This system is composed of two parts : an exposed system (and I assure you, I'm not happy about it), accessible through SFTP. (diode_src) a sealed system (as "lead and concrete box" sealed), commanding a ballistic system. (diode_dst)


These systems are connecting using a network diode. And diode_dst can only be controlled using archives pushed from diode_src.

Aegis Tech have "lost control" of SAFE a few weeks ago, and hadn't reached about it. If you find any information about the timeline of events and what the heck happen, please add it to your report. However that's not your main task : the exposed system has been tempered with, and archive sent to assess status and take back control on SAFE are not transmitted. Can you investigate and fix the issue ?

It seems that one of their admin investigated, in the early stages of the incident, some suspicious crashes, it may be a good starting point.

1 Reconnaissance

Nous avons accès à un serveur en SFTP :

 flag.txt	71	Document ...	31/03/2026 19:20:34	-rw-----
 log		Dossier de ...	13/04/2026 18:08:40	drwxr-xr-x
 archive		Dossier de ...	13/04/2026 15:15:32	drwxr-xr-x
 in		Dossier de ...	13/04/2026 15:15:08	drwxrwxr-x

- in : répertoire ouvert en écriture
- log : journaux des traitements des fichiers reçus
- archive :
- flag.txt

2 Reconstruction

Il va s'agir d'analyser un dump. La résolution des symboles à l'exécution fait que les noms des fonctions ne sont plus disponibles lors du chargement dans *Ghidra*.

La commande :

```
eu-unstrip -n --core 260302_core
```

nous permet de savoir quelles sont les bibliothèques chargées :

```
0x55bbd94b5000+0x8000
ac1c544053e3e406cfff1295f1995efb2276720@0x55bbd94b5380 . - /home/
diode/diode_src
```

```

0x7f172ec46000+0x1f6000
    def5460e3cee00bfee25b429c97bcc4853e5b3a8@0x7f172ec463b8 . - /usr/lib/
    x86_64-linux-gnu/libc.so.6
0x7f172ee68000+0x2000
    cd8bd85d6541a77d1d5d534cc64314e3e69ea1dc@0x7f172ee685c0 . - linux-
    vdso.so.1
0x7f172ee6a000+0x38000
    da08404553b441511cbec6b34bc78fe29fd5d14c@0x7f172ee6a248 . - /usr/lib/
    x86_64-linux-gnu/ld-linux-x86-64.so.2
0x7f172ee3c000+0x23010 808
    e7d31cb4401908ad687d5a935d533f9b3e144@0x7f172ee3c248 /lib/x86_64-
    linux-gnu/liblzo2.so.2 - liblzo2.so.2

```

En utilisant *pwntools* et le *build-id* :

```
pwn libcdb hash -t buildid def5460e3cee00bfee25b429c97bcc4853e5b3a8
```

nous récupérons la même *libc* que celle qui était chargée en mémoire lors du dump.

Mon objectif était ensuite d'utiliser directement les noms des fonctions pour l'analyse de *260302_core*. Mais je n'ai pas réussi à automatiser l'import (que ce soit via la génération de FID ou via l'import de la lib).

Dans un premier temps, nous utilisons *Ghidra* pour charger la *libc.so.6* et via *Memory Map*, *Set Image Base* nous la plaçons à la même adresse mémoire que celle à laquelle elle se trouve dans le dump (0x7f172ec46000).

Dans une autre fenêtre, nous ouvrons *260302_core* et "réparons" à la main la *got.plt* qui se trouve aux adresses 0x55bbd94bc000-0x55bbd94bcfff.

On sait que les adresses 7f172ecXXXX correspondent à des fonctions de la *libc*. On retrouve leurs noms en comparant les adresses aux fonctions se trouvant à ces adresses dans la *libc.so.6*.

Les adresses en 0x55bbd94bcXXX correspondent sans doute aux fonctions pas encore résolues par le lazy-binding.

On peut ainsi renommer les stubs de la .plt (0x55bbd94b7000 - 0x55bbd94b9fff).

Cela nous permet d'y voir un peu plus clair dans la retro-ingénierie du programme.

3 Que fait *diode_src* ?

Grâce à *gdb* :

```
gdb -c 260302_core
```

nous apprenons quelles fonctions sont incriminées dans le crash :

```

Core was generated by '/home/diode/diode_src'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0 0x00007f172edaa3c9 in ??()
(gdb) bt
#0 0x00007f172edaa3c9 in ??()
#1 0x000055bbd94b89aa in ??()
#2 0x0000000000000000 in ??()

```

Pour résumer, le programme surveille le répertoire "data/in" (fonction 0x55bbd94b7526). À la réception d'un nouveau fichier, le programme appelle la fonction de traitement :

```

void ProcessFile(long filename_path)
{
    int result;
    void *__ptr;
    package *memptr;
    long isArchive;

    if (filename_path != 0) {
        __ptr = (void *)CreateLogFile(filename_path);
        OpenLogFile(__ptr,1);
        LogInfo(ptrLogFile,1,"Processings_\%s",filename_path);
        memptr = (package *)arch_init_parser(filename_path);
        if (memptr == (package *)0x0) {
            LogInfo(ptrLogFile,2,"arch_init_parser()\_failed");
        }
        else {
            LogInfo(ptrLogFile,1,"Archive\_mapped\_at\_%p,\_size\_is\_%zu",memptr->
                mmap_base,memptr->fileSize);
            result = CheckCRC(memptr);
            if (result == 0) {
                LogInfo(ptrLogFile,1,"CRC\_is\_valid");
                result = arch_parse((package2 *)memptr);
                if (result == 0) {
                    result = arch_decompress_pkg((package2 *)memptr);
                    if (result == 0) {
                        result = pkg_parse((package2 *)memptr);
                        if (result == 0) {
                            if (memptr[1].debug_mode != '\0') {
                                ExecuteFct((char *)memptr);
                            }
                            isArchive = SearchNodeTag((NodeList *)((long)&memptr[1].
                                fileSize + 2),"ARCHIVE");
                            if ((isArchive == 0) ||
                                (result = archive_file(memptr,filename_path,"data/
                                    archive"), result == 0)) {
                                result = CheckFlagSecret((package2 *)memptr);
                                if (result == 0) {
                                    result = arch_transfert_file(memptr,"10.0.55.150",0
                                        x6fd);
                                    if (result != 0) {
                                        LogInfo(ptrLogFile,2,"arch_transfert_file()\_failed")
                                            ;
                                    }
                                }
                                else {
                                    LogInfo(ptrLogFile,2,"check_secret()\_failed");
                                }
                            }
                            else {
                                LogInfo(ptrLogFile,2,"archive_file()\_failed");
                            }
                        }
                    }
                    else {
                        LogInfo(ptrLogFile,2,"pkg_parse()\_failed");
                    }
                }
            }
        }
    }
}

```

```

        else {
            LogInfo(ptrLogFile, 2, "arch_decompress_pkg() failed");
        }
    }
    else {
        LogInfo(ptrLogFile, 2, "arch_parse() failed");
    }
}
else {
    LogInfo(ptrLogFile, 2, "arch_check_crc() failed");
}
}
if (memptr != (package *)0x0) {
    FUN_55bbd94b7efa(memptr);
}
if (__ptr != (void *)0x0) {
    cfree(__ptr);
}
if (filename_path != 0) {
    thunk_FUN_55bbd94b7066(filename_path);
}
return;
}
return;
}

```

3.1 Structure du fichier

La fonction *arch_parse* (0x55bbd94b8014) nous en apprend beaucoup sur la structure attendue du fichier :

Offset	Taille	
0x8	0x8	CRC
0x10	0x8	offsetPkg
0x18	0x8	szDecompress
0x20	0x8	offsetSig
0x28	0x8	offsetSecret
0x30	?	Tags
offsetPkg	(offsetSig - offsetPkg)	Package
offsetSig	(offsetSecret - offsetSig)	Signature
offsetSecret	(fileSz - offsetSecre)	Secret

Un calcul de CRC est réalisé sur le contenu du fichier à partir de 0x10.

3.1.1 Tags

Il s'agit d'une suite de chaînes terminées par des null.

Le tag "DEBUG" augmente la verbosité de la journalisation.

Le mode debug permet aussi l'utilisation du tag "_SHA256" qui a pour effet de réaliser un hash du fichier.

3.1.2 Package

En utilisant la même démarche que pour la *libc*, avec la *liblzo2*, on se rend compte qu'un appel est fait à *lzo1x_decompress_safe* afin de décompresser cette section du fichier. Une vérification est faite pour s'assurer que la taille du résultat correspond bien à la taille annoncée dans l'entête.

Des vérifications sont ensuite faites sur la structure des données décompressées :

Offset	Taille	
0x0	0x4	"MCRY"
0x4	0x4	Nb blobs
0x0	0x4	"AKNG"
0x4	0x4	szBlob
0x8	0x4	Type
0xc	szBlob	data
...

3.1.3 Signature

Pas d'information. C'est l'objet du Step suivant.

3.1.4 Secret

Cette partie du fichier est comparée à la valeur contenue dans "data/flag.txt". Information intéressante, le programme a donc des droits sur le fichier.

4 Exploitation

Revenons sur la possibilité de réaliser un hash du fichier :

```
void ExecuteFct(char *filename)
{
    char *__s;
    long result;
    void *__ptr;

    __s = malloc(0x1000);
    if (__s != (char *)0x0) {
        result = SearchNodeTag((NodeList *) (filename + 0x60), "_SHA256");
        if (result != 0) {
            snprintf(__s, 0x1000, "sha256sum_%s", *(undefined8 *) filename);
            __ptr = (void *) ExcuteFile(__s);
            if (__ptr != (void *) 0x0) {
                LogInfo(ptrLogFile, 0, "%s returned:\n%s", __s, __ptr);
                cfree(__ptr);
            }
        }
        if (__s != (char *) 0x0) {
            cfree(__s);
        }
    }
    return;
}
```

```
void * ExcuteFile(undefined8 commande)
{
    void *__ptr;
    long fp;

    __ptr = malloc(0x400);
}
```

```

if (__ptr == (void *)0x0) {
    __ptr = (void *)0x0;
}
else {
    fp = popen(commande,&idFile);
    if (fp == 0) {
        cfree(__ptr);
        __ptr = (void *)0x0;
    }
    else {
        fread(__ptr,1,0x400,fp);
        _closeFile(fp);
    }
}
return __ptr;
}

```

Si le tag `_SHA256` est trouvé, une chaîne est construite à partir de "sha256sum %s" et du nom du fichier. La commande est ensuite exécutée via *popen*.

Cela semble être un bon candidat pour une injection de commande.

Après plusieurs tentatives pour essayer de faire remonter le contenu du fichier dans le résultat de la commande, autre idée : faire passer une commande pour s'octroyer les droits en lecture sur le fichier.

Il ne reste plus qu'à créer un fichier répondant aux différentes contraintes + tag *DEBUG* et `_SHA256`. Et on le nomme : **test**; **chmod -R 777 data**.

On récupère ainsi le flag ce qui nous permet de compléter la section **Secret** dans notre script de création de fichier.

Nous pouvons maintenant créer des fichiers conformes aux vérifications de *diode_src*. Lorsqu'un fichier est valide, *diode_src* le transmet à *diode_dst*.

