Challenge SSTIC 2015

Vincent Bénony
bsr43@cryptic-apps.com

Cryptic Apps SARL

avril 2015

Table des matières

Table des figures

Chapitre 1

Stage 1 : La carte SD

La première étape de ce challenge consiste à analyser une carte SD, qui aurait été utilisée avec d'une clé USB *étrange*. Le fichier peut être téléchargé à cette adresse : `http://static.sstic.org/challenge2015/challenge.zip`.

Il s'agit d'un fichier ZIP, contenant un unique fichier *sdcard.img*. Le résultat de la commande `file` nous indique qu'il s'agit d'une image disque, au format FAT16 :

```
$ file sdcard.img
sdcard.img: x86 boot sector, mkdosfs boot message display, code offset 0
    x3c, OEM-ID "mkfs.fat", sectors/cluster 4, root entries 512, Media
    descriptor 0xf8, sectors/FAT 244, heads 64, sectors 250000 (volumes
    > 32 MB) , serial number 0xe50d883b, unlabeled, FAT (16 bit)
```

Voyons ce que cette image disque contient ; pour cela, il suffit la monter, grâce à la commande

```
$ mount -o loop,ro sdcard.img /mnt/img
```

À première vue, l'image ne contient qu'un seul fichier *inject.bin*, et cette fois-ci, la commande `file` n'est d'aucune utilité. Il va falloir creuser un peu plus pour comprendre ce que ce fichier contient. Je décide donc de regarder le contenu du fichier *sdcard.img* dans un éditeur hexadécimal, afin de savoir si des fichiers auraient été effacés de la carte.

1.1 Analyse des données FAT16

Le format FAT16 est assez rudimentaire. Les données sont regroupées en secteurs, et l'ensemble est constitué de 4 grandes parties placées les unes à la suite des autres :

- le secteur de démarrage (Boot Sector),
- la table d'allocation des fichiers (FAT),
- le répertoire racine (Root Directory),
- la zone des données.

1.1.1 Le Boot Sector

Afin de connaître la taille d'un secteur, il suffit de lire 2 octets (Little Endian) à l'offset 0xB ; ici, un secteur occupe 512 octets. La commande `file` nous avait déjà renseigné sur la taille de la FAT (244 secteurs), soit $244 \times 512 = 124\ 928$ octets. La FAT se situant juste après le secteur de démarrage, il suffit maintenant de regarder quelle est la taille de ce secteur. L'information se situe à l'offset 0xE (Reserved Sectors), où nous pouvons lire qu'il occupe 1 secteur, soit 512 octets. La FAT se situe donc à l'offset 0x200. Toujours dans les données du Boot Sector, nous apprenons qu'il y a 2 FATs (offset 0x10).

Chaque fichier stocké sur un périphérique formaté en FAT16 est découpé en cluster. Un cluster est un ensemble de secteurs (1 cluster = 4 secteurs dans notre cas). La FAT sert à distinguer quels sont les clusters utilisés pour stocker des données, de ceux qui sont libres. Quand un cluster est utilisé pour stocker des données, la FAT permet de connaître le numéro de secteur suivant. C'est le Root Directory qui nous renseigne sur le numéro du premier secteur utilisé pour un fichier, ainsi que la taille totale du fichier, et différents attributs.

Le Root Directory se trouve tout juste après les 2 FATs. Son offset est donc $512 + 2 \times 124\ 928 = 250\ 368 =$ `0x3D200`.

1.1.2 Le Root Directory

Le Root Directory contient la liste des fichiers que l'on trouve à la racine du périphérique. Il s'agit d'une simple liste d'objets de 32 octets, structurés ainsi :

Offset	Taille	Description
`0x00`	8 octets	nom du fichier
`0x08`	3 octets	extension
`0x0B`	1 octets	attributs
`0x0C`	1 octets	reservé
`0x0D`	1 octets	heure de création (ms)
`0x0E`	2 octets	heure de création
`0x10`	2 octets	date de création
`0x12`	2 octets	date du dernier accès
`0x14`	2 octets	reservé pour FAT32
`0x16`	2 octets	heure de la dernière écriture
`0x18`	2 octets	date de la dernière écriture
`0x1A`	2 octets	numéro du premier cluster
`0x1C`	4 octets	taille du fichier en octets

Table 1.1 – Un entrée du Root Directory

Quand un fichier est supprimé du périphérique, ses données ne sont pas réellement effacées. À la place, les clusters qui étaient occupés sont marqués comme étant libres, et son entré dans le Directory parent est modifié de telle sorte à ce que le premier caractère de son nom soit remplacé par la valeur `0xE5`.

```
003D200: e562 0075 0069 006c 0064 000f 00bd 2e00  .b.u.i.l.d......
003D210: 7300 6800 0000 ffff ffff 0000 ffff ffff  s.h.............
003D220: e555 494c 4420 2020 5348 2020 0000 2d1e  .UILD   SH  ..-.
003D230: 7a46 7a46 0000 2d1e 7a46 0300 2d00 0000  zFzF..-.zF..-...
003D240: 4169 006e 006a 0065 0063 000f 00e4 7400  Ai.n.j.e.c....t.
003D250: 2e00 6200 6900 6e00 0000 0000 ffff ffff  ..b.i.n.........
```

```
003D260: 494e 4a45 4354 2020 4249 4e20 0000 3a1e  INJECT   BIN  ..:.
003D270: 7a46 7a46 0000 3a1e 7a46 0400 a2ab 0a02  zFzF..:.zF......
003D280: 0000 0000 0000 0000 0000 0000 0000 0000  ................
003D290: 0000 0000 0000 0000 0000 0000 0000 0000  ................
003D2a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
003D2b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Nous voyons bien apparaître le fichier *INJECT.BIN*, mais il semble qu'il reste des traces de deux autres fichiers. Malheureusement, le premier fichier est totalement inutilisable (la plupart des champs ont été visiblement écrasés par des `0x00` ou des `0xFF`). Par contre, nous avons toujours l'information exploitable concernant le deuxième fichier : son nom était visiblement *BUILD.SH*, il contenait `0x2D` = 45 octets, et il commençait au cluster numéro 3. La zone des données des fichiers commence juste après le Root Directory, qui contient au plus 512 entrées (cette information est disponible offset `0x11` du Boot Sector). La zone des données se trouve donc à l'offset $\texttt{0x3D200} + 512 * 32 = 0x41200$. Sachant que les deux premiers clusters sont réservés, pour connaître l'offset du premier fichier, il suffit de calculer $\texttt{0x41200} + (3-2) * (512 * 4) = \texttt{0x41A00}$.

```
0041A00: 6a61 7661 202d 6a61 7220 656e 636f 6465  java -jar encode
0041A10: 722e 6a61 7220 2d69 202f 746d 702f 6475  r.jar -i /tmp/du
0041A20: 636b 7973 6372 6970 742e 7478 7400 0000  ckyscript.txt...
0041A30: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

En voilà une belle indication ! Mais surtout, en voilà une première belle frustration : il s'agissait d'une chaîne de caractères, qui aurait pu être facilement découverte avec la commande :

```
$ strings sdcard.img
        snip...
A A!A"A#A$A%A&A'A(A)A*A+A,A-A.A/A0A1A2A3A4A5A6A7A8A9A:A;A<A=A>A?A\
    @AAABACADAEAFAGAHAIAJAKALAMANAOAPAQARASATAUAVAWAXAYA
UILD    SH
zFzF
INJECT   BIN
zFzF
java -jar encoder.jar -i /tmp/duckyscript.txt
```

Malgré tout, nous avons ici une information capitale sur le type du fichier *inject.bin* : il s'agit d'un fichier Duckyscript compilé.

1.2 Décodage du Duckyscript

La clé USB étrange mentionnée dans la description du challenge est en fait une clé USB Rubber Ducky. Il s'agit d'une clé programmable, qui se fait passer pour un clavier auprès du système d'exploitation de l'ordinateur auquel elle est connectée.

Elle se programme à l'aide d'un langage de script assez simple, contenant des instructions d'attente (`DELAY`), des chaînes de caractères à simuler (`STRING`) ou des combinaisons de touches à presser (`WINDOWS`, `MENU`, *etc.*).

Un compilateur peut être téléchargé sur GitHub à l'adresse `https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Duckyscript`. Malheureusement, je

n'ai pas trouvé de "décompilateur" (mais je n'ai pas beaucoup cherché non plus...).

Le format est très simple : le fichier est une suite de paquets de 2 octets, correspondants aux codes des touches à presser, ou une valeur spéciale pour introduire un délai. Le décodage ne présentant que très peu d'intérêt, le code source complet est donné en annexe A sans plus de détails.

Une fois le fichier *inject.bin* décodé, nous obtenons un fichier de la forme :

```
DELAY 255
DELAY 255
DELAY 255
DELAY 255
DELAY 255
DELAY 255
DELAY 255
DELAY 215
WINDOWS r
DELAY 255
DELAY 245
ENTER
DELAY 255
DELAY 255
DELAY 255
DELAY 235
STRING cmd
ENTER
DELAY 50
STRING powershell
SPACE
STRING -enc
SPACE
STRING ZgB1AG4AYwB0AGkAbwBuACAAdwByAGkAdABlAF8AZgBpAGwAZQBfAGIAeQB0AGUA
cwB7AHAAYQByAGEAbQAoAFsAQgB5AHQAZQBbAF0AXQAgACQAZgBpAGwAZQBfAGIAeQB0AGU
AcwAsACAAWwBzAHQAcgBpAG4AZwBdACAAJABmAGkAbABlAF8AcABhAHQAaAAgAD0AIAAiAC
4AXABzAHQAYQBnAGUAMgAuAHoAaQBwACIAKQA7ACQAZgAgAD0AIABbAGkAbwAuAGYAaQBsA
GUAXQA6ADoATwBwAGUAbgBXAHIAaQB0AGUAKAAkAGYAaQBsAGUAXwBwAGEAdABoACkAOwAk
AGYALgBTAGUAZQBrACgAJABmAC4ATABlAG4AZwB0AGgALAAwACkAOwAkAGYALgBXAHIAaQB
0AGUAKAAkAGYAaQBsAGUAXwBiAHkAdABlAHMALAAwACwAJABmAGkAbABlAF8AYgB5AHQAZQ
BzAC4ATABlAG4AZwB0AGgAKQA7ACQAZgAuAEMAbABvAHMAZQAoACkAOwB9AGYAdQBuAGMAd
ABpAG8AbgAgAGMAaABlAGMAawBfAGMAbwByAHIAZQBjAHQAXwBlAG4AdgBpAHIAbwBuAG0A
ZQBuAHQAewAkAGUAPQBbAEUAbgB2AGkAcgBvAG4AbQBl snip...
DELAY 10
ENTER
STRING powershell
SPACE
snip...
```

Il s'agit d'un motif qui se répète en boucle : la clé attend un peu, simule l'appui simultané des touches Windows + R, saisi la chaîne de caractère "cmd", puis appuie sur la touche Enter. Sur une machine Windows, cette séquence à pour but de lancer un *invite de commande*.

Ensuite, le script y exécute une commande de la forme `powershell -enc BASE_64_STRING`.

Tous les scripts PowerShell exécutés suivent un même format. Une fois le BASE64 décodé, ils ressemblent à ceci :

```
function write_file_bytes{
        param(
                [Byte[]] $file_bytes,
                [string] $file_path = ".\stage2.zip");
        $f = [io.file]::OpenWrite($file_path);
        $f.Seek($f.Length,0);
        $f.Write($file_bytes,0,$file_bytes.Length);
        $f.Close();
```

```
}

function check_correct_environment{
        $e=[Environment]::CurrentDirectory.split("\");
        $e=$e[$e.Length-1]+[Environment]::UserName;
        $e -eq "challenge2015sstic";
}

if(check_correct_environment){
        write_file_bytes([Convert]::FromBase64String(
        'UEsDBAoDAAAAADaKeUbfS/XdEKUHABClBwAJAAAAZW5jcnlwdGV
        kfL/V3iijvVZtKEk8fNyka1PvkbrI0KkNNLxpo0np9mik2+VSO8D
        pYpziIGfM8vObQ6eCnQeaSrcOE9Lp9sRkxSyqYp/V1tjWu5V1NeJ
        s4Vom37rm7Pdoym+SoGdEyrFay1pbK1rzSNAfJiLMvtDBPZSwPcY
        nDg6Irh1U5U+o5MmS+LhV5LFRRTqcbi8INTNCfig/l1xFdK+/wFG
        ai22ni90MkzxzzOrV+09nBCHnm1+h5CAr2foL9oe8e11GNNKZt1m
        Umf4OB+/kw93C+Bya4HCBjJ3n7v+fBTwZq3D6BrlqRS1cnyZsEGl
        lLopvLxJUSMwUlL+uD2VBoaRodD/ZCYh2YEMRtTgNF+pN/2O2lQK
        79RsyO6EQL6RfejxdPYsPxqvPfw7UsnfjLydxKoPLPMOSxORYo1t
        cdh81ozeR4Iv3ZaYFUJlTFYSWpCBQtZFEuQfhdUTsVTCf0LcVG7f
        FtMxG9mwr+kez7Wt7VyANSQsKzxiSEU+2x+xAG0A1U5rDUqOOhdu
        pHI1+TLggOHlbHP2KKbW0dYtRYKomNEWhApdZWhHSU+QF2KYhFYD'
                snip...
        ));
}
else{
        write_file_bytes([Convert]::FromBase64String('
            VAByAHkASABhAHIAZABlAHIA')));
}
```

Ce script PowerShell ouvre un fichier *stage2.zip* en écriture, et y ajoute des données à la fin. J'ai donc décidé d'écrire un script Python qui collecte tous les scripts PowerShell, et qui extrait les données que le script est censé ajouter au fichier *stage2.zip*, dans le but de le reconstruire sans PowerShell. Le listing est donné en annexe B. Le fichier obtenu est bien un fichier ZIP valide, nous pouvons passer au Stage 2.

Chapitre 2

Stage 2 - OpenArena

Une fois le fichier *stage2.zip* décompressé, nous nous retrouvons avec 3 fichiers, *encrypted*, *memo.txt* et *sstic.pk3*. Le fichier *memo.txt* nous donne quelques indications, comme le nom d'un système de chiffrement (l'AES en mode OFB), ainsi qu'un vecteur d'initialisation. Il faut donc retrouver la clé de chiffrement AES qui a été utilisée pour chiffrer le fichier *encrypted*.

2.1 Carte OpenArena

Une fois encore, la commande `file` va nous donner quelques précisions sur la nature du dernier fichier.

```
$ file sstic.pk3
sstic.pk3: Zip archive data, at least v2.0 to extract
```

Ce fichier ZIP contient un grand nombre de fichiers, dont un fichier *README* qui donne quelques indications supplémentaires sur la manière d'utiliser le fichier PK3. Il s'agit en fait d'un niveau pour le jeu OpenArena, que je m'empresse donc d'installer. N'ayant jamais joué à ce type de jeu, je suis les indications fournies par le fichier *README*, et je sélectionne la carte SSTIC, après avoir cherché un bon moment sur Internet, la touche à utiliser sous OS X pour ouvrir la console, mais c'est un autre problème... (figure 2.5)

Je me balade, non sans mal, dans le niveau, et je remarque qu'à différents endroits, des quads contenant des chaînes hexadécimales sont présents. Après quelque temps à visiter le niveau, je remarque la présence d'interrupteurs (figure 2.2).

C'est là que commence mon calvaire : le principe est d'aller tirer sur l'interrupteur, atteindre une autre pièce en moins de 15 secondes, appuyer sur un autre interrupteur, revenir dans la pièce de départ, pour activer un panneau, ce qui nous envoie dans un autre endroit. J'arrive près d'un grand obstacle, que je suis censé franchir à l'aide d'un *rocket jump*[1] (figure 2.3). Après plusieurs tentatives, force est de constater que ce type d'exercice n'est pas pour moi, et que je n'y arriverai jamais... Il est temps de procéder autrement...

1. Le rocket jump est une technique de saut qui consiste à tirer une rocket au sol pour bénéficier de l'effet de l'explosion, afin de sauter plus loin...

FIGURE 2.1 – La carte SSTIC dans OpenArena

2.2 Tricher dans OpenArena

La carte aillant dut être éditée par les auteurs du challenge, et OpenArena étant un projet Open Source, j'en conclus qu'il doit être possible d'explorer la carte autrement, grâce à un éditeur. Après quelques recherches, je découvre GTKRadiant [2]. Étant donné que dans GTKRadiant, il y a GTK, je m'empresse d'oublier l'idée de lancer l'éditeur sous OS X, et j'opte plutôt pour la version Linux.

La première étape, pour éditer la carte, est de transformer le fichier *bsp*, en un fichier *map*. Pour ce faire, nous allons utiliser la commande `q3map2` fournie avec GTKRadiant.

```
$ q3map2 -game quake3 -fs_game baseoa -fs_basepath ~/sstic2015/
    oa/baseq3 -convert -format map ~/sstic2015/oa/maps/sstic.
    bsp
```

La commande génère un fichier *sstic_converted.map* qui peut être ouvert dans GTKRadiant2.4. Le processus n'est malheureusement pas parfait, car il va manquer beaucoup d'informations dans le fichier *map*, mais il y en aura assez pour résoudre cette partie du challenge.

La carte est constituée de trois zones (figure 2.5). La première, et la plus

2. `http://icculus.org/gtkradiant/`

FIGURE 2.2 – Un des interrupteurs

grande, est celle dans laquelle le joueur arrive au tout début de la partie, et celle qui contient les interrupteurs mentionnés plus tôt. La zone à sa gauche est celle du *rocket jump*. Enfin, la zone en haut à gauche est une zone qui n'a pas encore été explorée.

En lisant un peu de documentation sur OpenArena, je découvre qu'il existe différentes commandes qui vont m'être utiles pour explorer la carte comme bon me semble.

- La première d'entre elles est la commande `\devmap` : elle va permettre d'utiliser le mode développeur, et débloquer toutes les commandes qui suivent.
- La commande `\music` va permettre de rester *zen* pendant que je travaille sur cette carte.
- La commande `\g_gravity` permet de changer l'intensité de la force de gravité, et ainsi faire des sauts plus importants.
- Enfin, la commande `\setviewpos` va me permettre de me téléporter à n'importe quel endroit sur la carte.

En regardant le fichier *map* généré, je découvre la chaîne `"Yes!\n You found my key!"`. Je récupère les coordonnées de la zone, et dans le jeu, j'entre la commande :

```
\setviewpos -2388 2357 -424 270
```

FIGURE 2.3 – Rocket jump

J'arrive dans une pièce, dans laquelle se trouve un message qui m'indique la marche à suivre pour reconstruire la clé (figure 2.6).

Le principe est le suivant : dans le niveau de départ se trouvent des inscriptions, et sur chaque groupe d'inscriptions se trouve un petit symbole (un drapeau, une onde, une goutte, *etc.*). Il faut se balader dans le niveau, et noter les chaînes hexadécimales. Ensuite, il suffit de reformer la clé en concaténant chaque partie dans l'ordre de l'image 2.6.

Je jette un œil au répertoire des textures, mais je m'aperçois très vite qu'il y a beaucoup de fausses textures, et que je vais devoir trouver lesquelles sont effectivement affichées dans le jeu. Je retourne dans le fichier *map*, et je filtre le contenu pour ne conserver que les endroits où sont utilisées les textures du répertoire *sstic*.

```
$ grep sstic sstic_converted.map | sort -u

( -1040.000 -564.000 -417.000 ) ( -1040.000 -564.000 -425.500 ) (
    -1040.000 -572.500 -425.500 ) sstic/02 0 0 0 0.5 0.5 0 0 0
( -1040.311 -697.651 -503.750 ) ( -1040.689 -574.401 -503.750 ) (
    -1040.689 -574.401 -379.250 ) sstic/52809216 0 0 0 0.5 0.5 0 0 0
( -1040.500 -572.500 -417.000 ) ( -1040.500 -572.500 -425.500 ) (
    -1040.500 -564.000 -425.500 ) sstic/02 0 0 0 0.5 0.5 0 0 0
( -1317.151 -567.289 -348.456 ) ( -1413.651 -567.430 -348.194 ) (
    -1413.651 -654.793 -395.294 ) sstic/457615433 0 0 0 0.5 0.5 0 0 0
( -2051.154 2379.609 -144.506 ) ( -2147.651 2379.311 -144.503 ) (
    -2147.649 2378.313 -243.748 ) sstic/103336131 0 0 0 0.5 0.5 0 0 0
( -2058.404 1691.704 -144.507 ) ( -2154.901 1691.407 -144.504 ) (
```

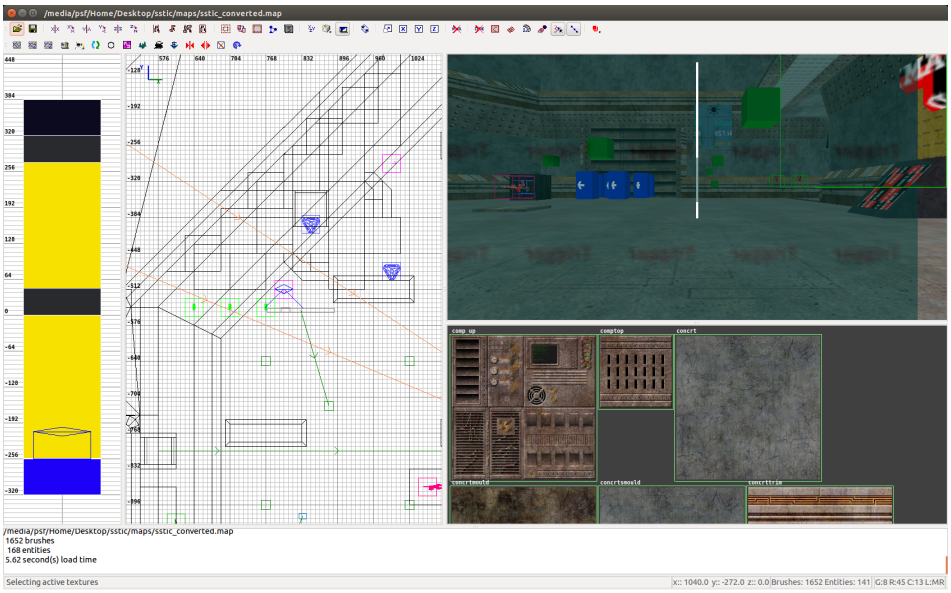


FIGURE 2.4 – L'éditeur GTKRadiant

```
    −2154.899 1690.408 −243.749 ) sstic/522168825 0 0 0 0.5 0.5 0 0 0
( −2058.654 1837.204 −144.507 ) ( −2155.152 1836.907 −144.504 ) (
    −2155.149 1835.908 −243.749 ) sstic/2068870268 0 0 0 0.5 0.5 0 0 0
( −2059.904 2187.609 −144.506 ) ( −2156.401 2187.311 −144.503 ) (
    −2156.399 2186.313 −243.748 ) sstic/1239519434 0 0 0 0.5 0.5 0 0 0
( −2059.904 2267.609 −144.506 ) ( −2156.401 2267.311 −144.503 ) (
    −2156.399 2266.313 −243.748 ) sstic/1234125232 0 0 0 0.5 0.5 0 0 0
( −2060.154 1757.204 −144.507 ) ( −2156.651 1756.907 −144.504 ) (
    −2156.649 1755.908 −243.749 ) sstic/457671845 0 0 0 0.5 0.5 0 0 0
( −2061.404 2026.204 −144.507 ) ( −2157.902 2025.907 −144.504 ) (
    −2157.899 2024.908 −243.749 ) sstic/156761256 0 0 0 0.5 0.5 0 0 0
( −2062.154 1888.204 −144.507 ) ( −2158.651 1887.907 −144.504 ) (
    −2158.649 1886.908 −243.749 ) sstic/2061717677 0 0 0 0.5 0.5 0 0 0
( −2063.154 2086.204 −144.507 ) ( −2159.652 2085.907 −144.504 ) (
    −2159.649 2084.909 −243.749 ) sstic/994048089 0 0 0 0.5 0.5 0 0 0
( −2119.000 2404.750 −254.500 ) ( −2119.000 2404.750 −513.500 ) (
    −2768.250 2404.750 −513.500 ) sstic/643008245 0 0 0 0.5 0.5 0 0 0
( −2119.000 2406.250 −255.500 ) ( −2119.000 2406.250 −514.500 ) (
    −2119.000 2400.250 −514.500 ) sstic/643008245 0 0 0 0.5 0.5 0 0 0
( −2119.000 2418.250 −254.500 ) ( −2119.000 2418.250 −513.500 ) (
    −2119.000 2404.750 −513.500 ) sstic/643008245 0 0 0 0.5 0.5 0 0 0
( −2130.250 2401.750 −321.250 ) ( −2130.250 2401.750 −352.500 ) (
    −2199.000 2401.750 −352.500 ) sstic/1429015180 0 0 0 0.5 0.5 0 0 0
( −2150.500 2426.750 −257.500 ) ( −2150.500 2426.750 −288.750 ) (
    −2081.750 2426.750 −288.750 ) sstic/1371257909 0 0 0 0.5 0.5 0 0 0
( −2150.500 2426.750 −370.500 ) ( −2150.500 2426.750 −401.750 ) (
    −2081.750 2426.750 −401.750 ) sstic/72359428 0 0 0 0.5 0.5 0 0 0
( −2171.154 2375.859 −144.506 ) ( −2267.651 2375.561 −144.503 ) (
    −2267.649 2374.563 −243.748 ) sstic/1036082074 0 0 0 0.5 0.5 0 0 0
( −2178.404 1687.954 −144.507 ) ( −2274.901 1687.657 −144.504 ) (
    −2274.899 1686.658 −243.749 ) sstic/522248554 0 0 0 0.5 0.5 0 0 0
( −2178.654 1833.454 −144.507 ) ( −2275.152 1833.157 −144.504 ) (
    −2275.149 1832.158 −243.749 ) sstic/2070448105 0 0 0 0.5 0.5 0 0 0
```

snip ...

On s'aperçoit très vite que toutes les textures sont utilisées par la map, mais que la plupart d'entre-elles sont utilisées dans une zone géographique qui n'est

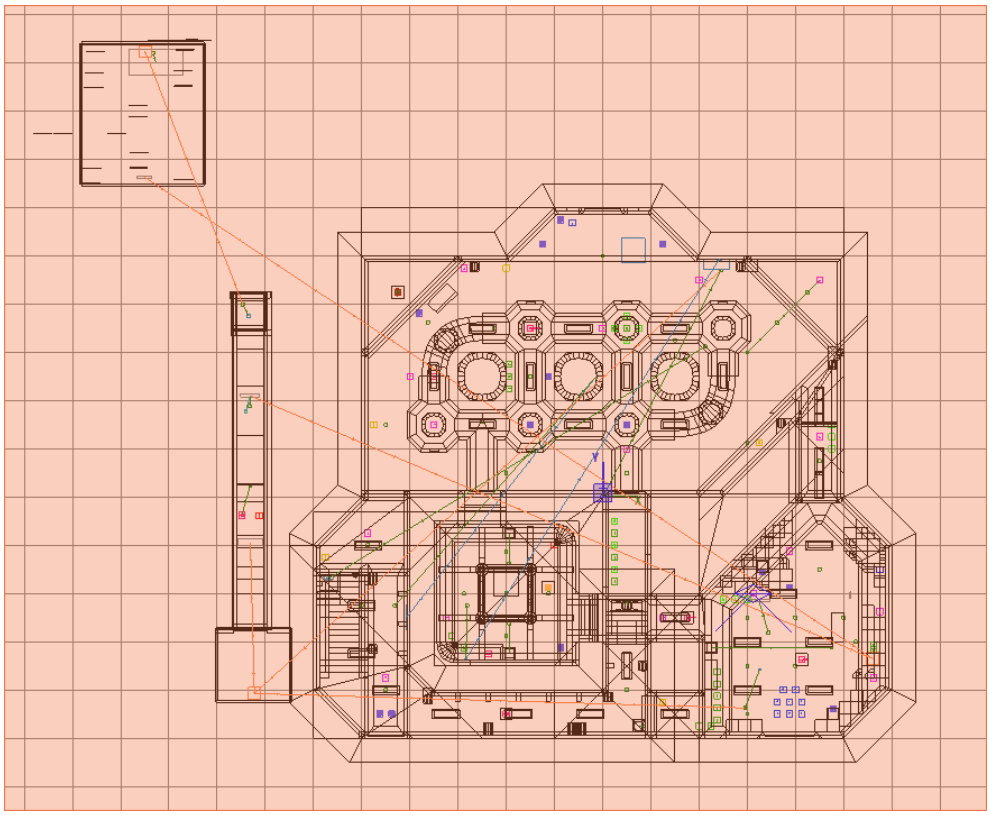


FIGURE 2.5 – Carte du niveau

pas accessible au joueur, en gros, dans la plage des $X \in [-2500; -2000]$. Il suffit donc de filtrer le résultat du grep.

Nous arrivons à ne conserver qu'un tout petit nombre de textures :

Symbole	Texture	Orange	Blanc	Vert
drapeau	457615433	b0daf152	db6e3063	9e2f31f7
onde	1773100136	8267d420	8153296b	12f7d028
boite	1604401524	3d9b0ba6	d07ccd3d	ca243465
goute	71921642	552f76e6	7695dc7c	3acad14b
chaîne	52809216	b00b7677	14e3ec8b	b54cdc34
wifi	86520831	2ce017fd	30c419d9	ffe0d355
ordinateur	838783866	795fbc7b	26609fac	4b763163

TABLE 2.1 – Liste des textures utilisées

Il est maintenant possible de reconstituer la clé de chiffrement :

`9e2f31f7 8153296b 3d9b0ba6 7695dc7c b0daf152 b54cdc34 ffe0d355 26609fac`

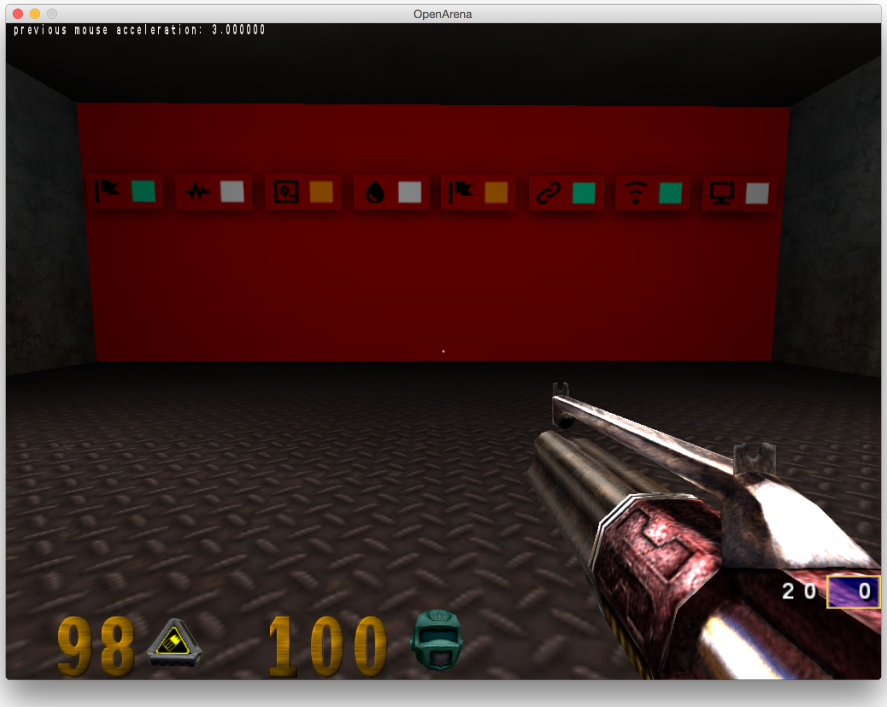


FIGURE 2.6 – La clé

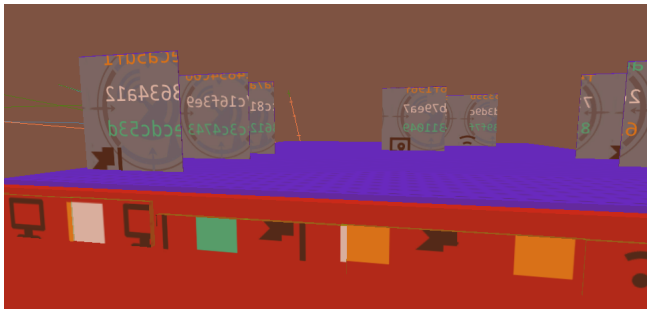


FIGURE 2.7 – Les fausses tuiles

Puis, de déchiffrer le fichier *encrypted* :

```
openssl enc -d -aes-256-ofb
  -K 9e2f31f78153296b3d9b0ba67695dc7cb0daf152b54cdc34ffe0d35526609fac
  -iv 53535449433230313 52d537461676532 -in encrypted -out decrypted.zip
```

Le fichier est bien un fichier ZIP valide, même si sa somme SHA256 ne correspond pas à celle du fichier *memo.txt*. Nous pouvons continuer à l'étape 3.

Chapitre 3

Stage 3 - Paint

Cette étape est certainement la plus simple des 6 étapes du challenge. Une fois le fichier ZIP décompressé, nous nous retrouvons avec trois fichiers *encrypted*, *memo.txt* et *paint.cap*.

3.1 Analyse de la trace USB

Encore une fois, un petit tour par la commande `file` :

```
$ file paint.cap
paint.cap: tcpdump capture file (little-endian) - version 2.4, capture
    length 262144)
```

Il s'agit donc d'une trace au format PCAP. Je décide de l'explorer avec WireShark (figure 3.1).

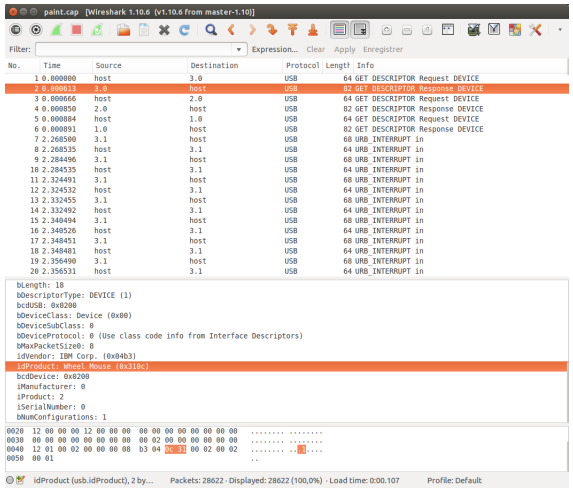


FIGURE 3.1 – Wireshark, et la trace PCAP

On découvre qu'il s'agit d'une trace de la connexion d'une souris USB, et de son utilisation.

3.2 Représentation graphique de la trace

Le protocole de communication d'une souris USB avec un ordinateur est très simple, elle envoie des paquets de 4 octets à chaque mouvement, suivant le format :

Offset	Description
0	toujours 0
1	delta X
2	delta Y
3	boutons

TABLE 3.1 – Format d'un paquet USB pour une souris

Il est donc possible d'écrire un programme qui dessine le chemin qu'à parcouru la souris lors de cette trace. J'ai décidé d'utiliser Xcode, et d'écrire un petit programme en Objective-C pour afficher le résultat. Le code de la vue de dessin est donné en annexe C.

Nous obtenons le dessin présenté figure 3.2.

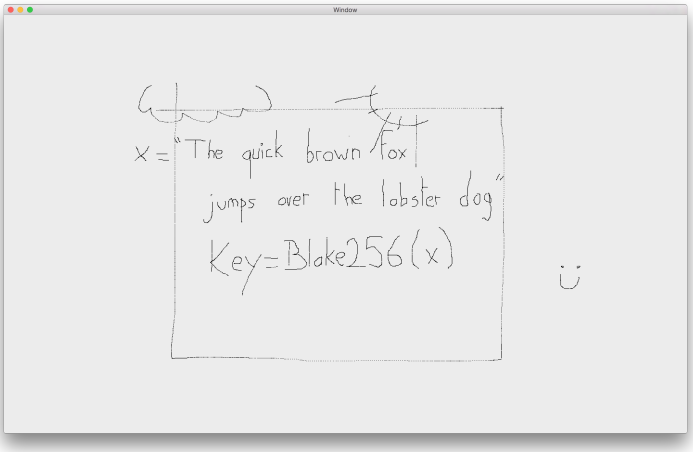


FIGURE 3.2 – Dessin à la souris

Pour calculer le hash BLAKE, j'utilise l'implémentation Python disponible à cette adresse : `http://www.seanet.com/~bugbee/crypto/blake/blake.py`.

```
#!/usr/bin/env python

from binascii import hexlify
from blake import BLAKE

msg = b'The quick brown fox jumps over the lobster dog'
hashlen = 256
digest = BLAKE(hashlen).digest(msg)
print hexlify(digest)
```

Et j'obtiens la clé

```
66c1ba5e8ca29a8ab6c105a9be9e75fe0ba07997a839ffeae9700b00b7269c8d
```

Maintenant, l'étape la plus "compliquée" : trouver une implémentation de l'algorithme Serpent-1. J'ai opté pour la bibliothèque Crypto++ [1]. Le code pour le déchiffrement se trouve en annexe D.

1. http://www.cryptopp.com

Chapitre 4

Stage 4 - JavaScript

Le Stage 4 est celui qui m'aura donné le plus de fil à retordre... et quand j'y repense, c'est malheureux, car c'est celui qui aurait dû être le plus simple si je l'avais compris dès le départ !

Le fichier ZIP contient un unique fichier HTML. Lorsqu'il est ouvert dans un navigateur, on obtient le message

Download manager

Failed to load stage 5

4.1 Analyse du JavaScript

En ouvrant le fichier dans un éditeur de texte, on remarque qu'il est essentiellement constitué d'un script JavaScript, qui déclare deux variables `data` et `hash`, suivies d'une série de caractères. Après une rapide recherche sur Internet, il apparaît que ces caractères sont issus d'un obfuscateur de code JavaScript qui s'appelle `jjencode`[1].

Afin de rendre le code lisible, j'ai décidé de placer un point d'arrêt sur la modification du DOM dans Chrome. Ainsi, dès que le `body` est modifié, Chrome arrêtera le JavaScript. C'est comme ça que l'on peut extraire une première version du script en partie nettoyée automatiquement par Chrome.

```
(function() {
    __ = document;
    $$$ = 'stage5';
    $$_$ = 'load';
    $_$$ = ' ';
    _$$$$$ = 'user';
    _$$$ = 'div';
    $$_$$$ = 'navigator';
    $$_$$ = 'preferences';
    $_$$$ = 'to';
    $$$_$ = 'href';
    $$$$_ = '=';
    $$$$$ = 'chrome';
    _$$$$ = '"';
    $_$$$$ = 'Agent';
```

1. `http://utf-8.jp/public/jjencode.html`

```
$$$_$$ = 'down';
$$$$_$ = 'import';
$ = '<b>Failed' + $_$$ + $_$$$ + $_$$ + $$_$ + $_$$ + $$$ + '</b>';
___ = 'write';
____ = 'getElementById';
_$_ = "raw";
$$ = window;
__$ = $$.crypto.subtle;
__$_ = 'decrypt';
___$ = 'status';
$____ = $$$$_$ + 'Key';
_______ = 0;
__$__ = 'then';
_$_____ = 'digest';
__$____ = 'innerHTML';
___$__ = {
    name: 'SHA-1'
};
____$_ = data;
_____$ = hash;
_$______ = Blob;
__$_____ = URL;
___$____ = 'createObjectURL';
_____$ = 'type';
$_______ = 'application/octet-stream';
_$______ = 'name';
__$_____ = 'AES-CBC';
___$____ = 'iv';
____$___ = '<a' + $_$$ + $$$_$ + $$$$_ + _$$$$;
_____$__ = '"' + $_$$ + $$$_$$ + $$_$ + $$$$_ + _$$$$ + $$$ + '.zip'
    + _$$$$ + '>' + $$$_$$ + $$_$ + $_$$ + $$$ + '</a>';
______$_ = '(';
______$$ = ')';
$________ = 'setTimeout';
_________ = parseInt;
__________ = $$[$$_$$$][_$$$$$ + $_$$$$];
___________ = 'length';
____________ = 'substr';
_____________ = _______ + 1;
______________ = _____________ * 2;
_______________ = ______________ * 4;
________________ = _____________ * _____________;
$_$ = 125 * _____________;
_________________ = 'indexOf';
__________________ = 'charCodeAt';
___________________ = 'push';
____________________ = Uint8Array;
_____________________ = '';
______________________ = 'byteLength';
_______________________ = $_$$$ + 'String';
__[___]('<h' + _____________ + '>Down' + $$_$ + $_$$ + 'manager</h'
    + _____________ + '>');
__[___]('<' + _$$$ + $_$$ + 'id' + $$$$_ + _$$$$ + ___$ + _$$$$ + '
    ><i>' + $$_$ + 'ing...</i></' + _$$$ + '>');
__[___]('<' + _$$$ + $_$$ + 'style' + $$$$_ + _$$$$ + 'display:none'
    + _$$$$ + '><a' + $_$$ + 'target' + $$$$_ + _$$$$ + 'blank' +
    _$$$$ + $_$$ + $$$_$ + $$$$_ + _$$$$ + $$$$$ + '://browser/
    content/' + $$_$$ + '/' + $$_$$ + '.xul' + _$$$$ + '>Back' +
    $_$$ + $_$$$ + $_$$ + $$_$$ + '</a></' + _$$$ + '>');

function _____(______) {
    _ = [];
    for (___________ = _______; ___________ < ______[___________];
        ++___________) _[___________________](______[
        ___________________](___________));
    return new ____________________(_);
}

function ____________________(______) {
    _ = [];
    for (___________ = _______; ___________ < ______[___________] /
        _____________; ++___________) _[___________________](
        _________(______[_____________](___________ *
        _____________, _____________), ___________________));
```

```
        return new _____________________(_);
    }

    function _____________________(________) {
        ______ = _____________________;
        for (___________ = _______; ___________ < ________[
            _____________________]; ++___________) {
            ___ = ________[___________][_____________________](
                ____________________);
            if (___[_____________] < _________________) ______ += _______;
            ______ += ___;
        }
        return ______;
    }

    function _____________________() {
        $_ = _____(__________[_____________](__________[
            ____________________](_______$_) + _________________,
            ____________________));
        _$__ = _____(__________[_____________](__________[
            ____________________](_______$) - _____________________,
            ____________________));
        _$ = {};
        _$[_$_______] = __$_____;
        _$[___$____] = $_;
        _$[_____________] = _$__[_____________] * ____________________;
        __$[$____](_$_, _$__, _$, false, [__$_])[__$__](function(_$___)
            {
            __$[__$_](_$, _$___, _____________________(____$_))[__$__](
                function(___$_) {
                ____$ = new _____________________(___$_);
                __$[_$____](___$__, ____$)[__$__](function(_____$$) {
                    if (_____$ == _____________________(new
                        _____________________(____$$))) {
                        _____$_ = {};
                        _____$_[_______$] = $________;
                        _____$ = new _$_____([____$], _____$_);
                        __$____ = ___$____[____$__](_____$);
                        __[____](___$)[__$___] = ____$___ + __$____ +
                            _____$__;
                    } else {
                        __[____](___$)[__$___] = $;
                    }
                });
            }).
            catch (function() {
                __[____](___$)[__$___] = $;
            });
        }).
        catch (function() {
            __[____](___$)[__$___] = $;
        });
    }
    $$[$________](_____________________, $_$);
})
```

Avec un peu de patience, un bon éditeur de texte comme TextMate, et quelques expressions régulières, il est possible de récupérer un JavaScript lisible :

```
(function() {
    $ = '<b>Failed to load stage5</b>';
    document.write('<h1>Download manager</h1>');
    document.write('<div id="status"><i>loading...</i></div>');
    document.write('<div style="display:none"><a target="blank" href="
        chrome://browser/content/preferences/preferences.xul">Back to
        preferences</a></div>');

    function stringToByteArray(variable) {
        r = [];
        for (iter = 0; iter < variable.length; ++iter) {
            r.push(variable.charCodeAt(iter));
```

```
        }
        return new Uint8Array(r);
    }

    function hexStringToArray(variable) {
        r = [];
        for (iter = 0; iter < variable['length'] / 2; ++iter) {
            r.push(parseInt(variable['substr'](iter * 2, 2), 16));
        }
        return new Uint8Array(r);
    }

    function bytesToHexString(arr) {
        variable = '';
        for (iter = 0; iter < arr.byteLength; ++iter) {
            w = arr[iter].toString(16);
            if (w.length < 2) variable += 0;
            variable += w;
        }
        return variable;
    }

    function func4() {
       var ua = window.navigator.userAgent;
       in_iv = stringToByteArray(ua.substr(ua.indexOf('(') + 1, 16));
       in_key = stringToByteArray(ua.substr(ua.indexOf(')') - 16, 16));

       cipherDesc = {
           'name': 'AES-CBC',
           'iv': in_iv,
           'length': in_key.length * 8
       }

        window.crypto.subtle.importKey("raw", in_key, cipherDesc, false,
              ['decrypt']).then(function(_$___) {
            window.crypto.subtle.decrypt(cipherDesc, _$___,
                hexStringToArray(data)).then(function(___$_) {
                uncipheredBytes = new Uint8Array(___$_);
                window.crypto.subtle.digest({ name: 'SHA-1' },
                    uncipheredBytes).then(function(____$$) {
                    if (hash == bytesToHexString(new Uint8Array(____$$))
                        ) {
                        hash = new Blob([uncipheredBytes], {
                            'type'] = 'application/octet-stream'
                        });
                        document.getElementById('status').innerHTML = '<
                            a href="' + URL['createObjectURL'](hash) +
                            '" download="stage5.zip">download stage5</a
                            >';
                    } else {
                        document.getElementById('status').innerHTML = $;
                    }
                });
            }).
            catch (function() {
                document.getElementById('status').innerHTML = $;
            });
        }).
        catch (function() {
            document.getElementById('status').innerHTML = $;
        });
    }
    window.setTimeout(func4, 1000);
})
```

Le script utilise la bibliothèque SubtleCrypto pour déchiffrer le contenu de la variable `data`. L'algorithme utilisé est de l'AES, en mode CBC. La clé, et le vecteur d'initialisation, sont dérivés du user agent du navigateur : le vecteur d'initialisation correspond aux 16 premiers caractères de la chaîne entre parenthèse dans le user agent, et la clé aux 16 derniers caractères.

Le but est donc de trouver un user agent qui permette de déchiffrer les données, sachant aussi que l'on dispose de la somme SHA1 des données déchiffrées.

En regardant d'un peu plus près le fichier HTML généré par le script, on remarque qu'un nœud `div` est présent, mais invisible. Ce nœud contient un lien :

```
<div style="display:none">
        <a target="blank"
            href="chrome://browser/content/preferences/preferences.xul">
                Back to preferences
        </a>
</div>
```

La référence au language d'interface graphique XUL nous laisse penser que le user agent à un rapport avec Firefox ; c'est en effet le seul navigateur à utiliser cette technologie.

4.2 Les user agent de Firefox

Les chaînes user agent de Firefox suivent un modèle assez précis. Le schéma est le suivant [2] :

Mozilla/5.0 (platform ; rv :geckoversion) Gecko/geckotrail Firefox/firefoxversion

- `Mozilla/5.0` est une chaîne qui se trouve toujours au début du user agent.
- `platform` correspond à l'OS. Il s'agit en général d'une chaîne comme `Macintosh`, `X11` ou `Windows NT 5.0`.
- `rv:geckoversion` correspond à la version du navigateur, comme par exemple `17.0`.
- `Gecko/geckotrail` est toujours égal à la chaîne `Gecko/20100101` sur ordinateur.
- `Firefox/firefoxversion` est le numéro de version de Firefox (identique au geckotrail).

Voici un exemple de user agent :

Mozilla/5.0 (Windows NT 3.1 ; rv :17.0) Gecko/20100101 Firefox/17.0

Il est possible d'écrire un script Python qui énumère tous les user agent possible, qui tente de déchiffrer les données, et qui vérifie que l'on obtient bien la somme SHA1 attendue.

```
#!/usr/bin/python

import os

bash_file = "cmds.sh"
output = open(bash_file, "w")
output.write("#!/bin/bash\n")

agents = []
```

2. `https://developer.mozilla.org/en-US/docs/Web/HTTP/Gecko_user_agent_string_reference`

```
win_nt_versions = ["3.1", "3.5", "3.51", "4.0", "5.0", "5.1" , "5.2", "
    6.0", "6.1", "6.2", "6.3", "10.0"]
win_versions = []
for v in win_nt_versions:
    win_versions.append("Windows NT " + v)

win_variation = ["", "; Win64", "; WOW64"]

def gen(output, iv, key):
    file = "data_stage4"
    openssl_cmd = "openssl enc -d -aes-128-cbc -K $(echo -n '" + key + "
        ' | xxd -ps) -iv $(echo -n '" + iv + "' | xxd -ps) -in " + file
    cmd = openssl_cmd + " 2>&1 >/dev/null && echo -n '" + iv + key + ":
        ' && " + openssl_cmd + " | shasum"
    output.write(cmd + "\n")

for version in win_versions:
    for variation in win_variation:
        ua = version + variation
        agents.append(ua)

for v in xrange(0, 11):
    agents.append("Macintosh; Intel Mac OS X 10.%d" % v)
    agents.append("Macintosh; PPC Mac OS X 10.%d" % v)

for agent in agents:
    for gecko in xrange(1, 37):
        ua = "%s; rv:%d.0" % (agent, gecko)
        iv = ua[:16]
        key = ua[-16:]
        gen(output, iv, key)

output.close()
os.system("bash " + bash_file + " | grep -v 'bad decrypt'")
```

Ce programme génère un script Bash, qu'il exécute ensuite. Au bout de quelques secondes, nous obtenons le couple IV = `"Macintosh; Intel"` et clé = `" X 10.6; rv:35.0"`.

Il est alors possible de modifier le user agent du navigateur, avec par exemple :

```
Mozilla/5.0 (Macintosh; Intel OS X 10.6; rv:35.0)
```

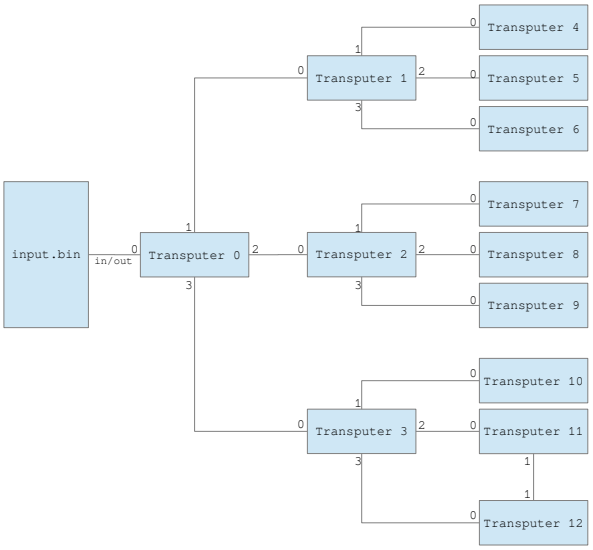
La page affiche maintenant un lien de téléchargement pour le fichier *stage5.zip*.

Chapitre 5

Stage 5 - Les transputers

Cette partie du challenge est certainement la plus amusante, et celle où j'ai appris le plus de choses.

Le fichier ZIP contient deux fichiers *input.bin* et *schematic.pdf*. Le fichier PDF décrit une architecture de transputers, comme ceci :




```
SHA256:
    a5790b4427bc13e4f4e9f524c684809ce96cd2f724e29d94dc999ec25e166a81 - encrypted
    9128135129d2be652809f5a1d337211affad91ed5827474bf9bd7e285ecef321 - decrypted

Test vector:
    key = "*SSTIC-2015*"
    data = "1d87c4c4e0ee40383c59447f23798d9fefe74fb82480766e".decode("hex")
    decrypt(key, data) == "I love ST20 architecture"
```

Je n'avais jamais entendu parler des transputers (honte à moi), mais le concept est séduisant : il s'agit d'un petit microprocesseur, pensé dès le départ pour être utilisé dans une architecture hautement parallélisée. Chaque CPU est

capable de démarrer depuis un flux (mémoire, réseau) suivant un protocole très simple. Le CPU reçoit une valeur N (sur un octet), il charge N octets depuis ce même flux qu'il copie en mémoire, puis l'exécute. Chaque CPU peut être connecté à d'autres CPU, et s'échanger entre eux des messages synchrones.

Il apparaît donc que le fichier *input.bin* contient des données permettant de démarrer l'architecture présentée sur le PDF. Il n'y a donc plus qu'à désassembler tout ça...

Pour ce faire, j'ai choisi d'écrire un plugin pour Hopper [1], dont le code source a été diffusé sur GitHub [2].

5.1 Le ST20

Le document PDF ne laisse aucun doute quant au CPU choisi : il s'agit du ST20 de STMicroelectronics.

Le jeu d'instruction est très réduit, et se décode avec beaucoup de facilité. Une grande partie du jeu d'instruction est codé sur un unique octet, qui est découpé en deux parties de 4 bits ; la partie haute représente l'opcode, et la partie basse une donnée pour l'instruction. Quand un octet ne suffit pas à enregistrer la donnée nécessaire, il est possible d'utiliser un prefix (opcode 0x20). De plus, afin d'étendre le jeu d'instruction, il est possible d'utiliser l'opcode d'échappement 0xF0.

5.2 Analyse dans Hopper

Une fois le plugin pour le ST20 installé, chargeons le fichier *input.bin* dans Hopper.

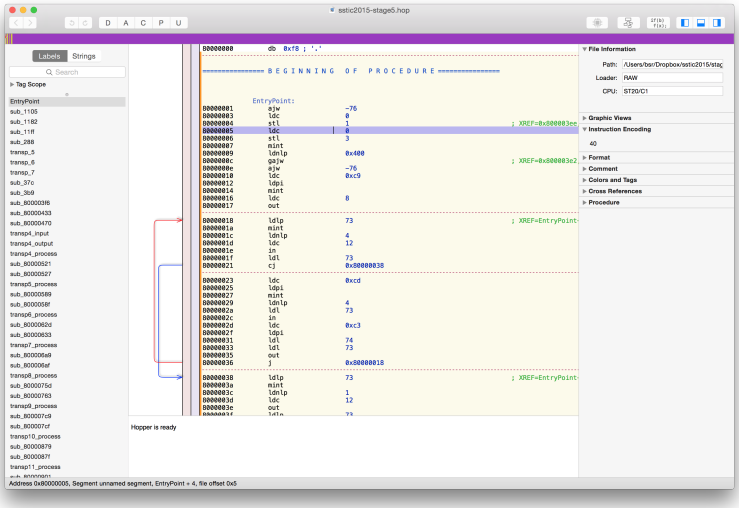


FIGURE 5.1 – Désassemblage ST20 dans Hopper

1. `http://www.hopperapp.com`
2. `https://github.com/bSr43/HopperST20C2C4-plugin`

Le premier octet du fichier correspond à la taille du boot code qui est envoyé au `Transputer 0`. Le programme envoie sur le port 0 les octets du message "Boot OK".

```
80000003          ldc                 0
80000004          stl                 1
80000005          ldc                 0
80000006          stl                 3
80000007          mint
80000009          ldnlp               0x400
8000000c          gajw
8000000e          ajw                 -76
80000010          ldc                 0xc9
80000012          ldpi                        ; 0x800000DD ("Boot OK")
80000014          mint
80000016          ldc                 8
80000017          out                         ; Affichage du message
```

Puis viens le code d'un dispatcher, qui va envoyer à chacun des transputer connectés au `Transputer 0`, le code dont il a besoin. Pour ce faire, le `Transputer 0` va lire 3 mots de 32 bits. Le premier mot est une longueur, le deuxième mot est l'adresse du canal où envoyer le code. Le troisième mot est inutilisé à cet endroit, mais ce format de paquet de 3 mots est quelque chose qui revient fréquemment dans le code.

```
              tr0_send_to_children:
80000018          ldlp                73
8000001a          mint
8000001c          ldnlp               4
8000001d          ldc                 12
8000001e          in                          ; Lecture de 12 octets

8000001f          ldl                 73
80000021          cj                  tr0_end       ; premier mot == 0?

80000023          ldc                 0xcd
80000025          ldpi                        ; Buffer en 0x800000F4
80000027          mint
80000029          ldnlp               4
8000002a          ldl                 73
8000002c          in                          ; Lecture boot code

8000002d          ldc                 0xc3
8000002f          ldpi
80000031          ldl                 74
80000033          ldl                 73
80000035          out                         ; envoi a l'enfant

80000036          j                   tr0_send_to_children
```

Grâce à cette première analyse rapide, nous apprenons comment le premier transputer sait ce qu'il doit envoyer comme code à exécuter aux autres transputers. J'ai écrit un petit programme en C qui va analyser le fichier *input.bin* afin de découvrir le début de chaque boot code de chaque transputer. Le code est très simple :

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

int main (int argc, char const *argv[])
{
```

```
    FILE *f = fopen("input.bin", "rb");

    fseek(f, 0xF9, SEEK_SET);

    while (1) {
        printf("offset 0x%lx: ", ftell(f));

        uint32_t header[3];
        fread(header, 4, 3, f);

        if (header[0] == 0) break;

        printf("taille = %d", header[0]);
        printf(", destination = transp_%d", (header[1] & 0xf) / 4);
        fseek(f, header[0], SEEK_CUR);
        puts("");
    }

    printf("fin a 0x%lx\n", ftell(f));

    return 0;
}
```

Nous obtenons ainsi l'offset dans le fichier du début de chaque boot code :

- offset `0xf9` : taille = 113, destination = $transp_1$
- offset `0x176` : taille = 113, destination = $transp_2$
- offset `0x1f3` : taille = 113, destination = $transp_3$
- offset `0x270` : taille = 49, destination = $transp_1$
- offset `0x2ad` : taille = 49, destination = $transp_1$
- offset `0x2ea` : taille = 49, destination = $transp_1$
- offset `0x327` : taille = 49, destination = $transp_2$
- offset `0x364` : taille = 49, destination = $transp_2$
- offset `0x3a1` : taille = 49, destination = $transp_2$
- offset `0x3de` : taille = 49, destination = $transp_3$
- offset `0x41b` : taille = 49, destination = $transp_3$
- offset `0x458` : taille = 49, destination = $transp_3$
- offset `0x495` : taille = 92, destination = $transp_1$
- offset `0x4fd` : taille = 92, destination = $transp_1$
- offset `0x565` : taille = 152, destination = $transp_1$
- offset `0x609` : taille = 112, destination = $transp_2$
- offset `0x685` : taille = 168, destination = $transp_2$
- offset `0x739` : taille = 96, destination = $transp_2$
- offset `0x7a5` : taille = 164, destination = $transp_3$
- offset `0x855` : taille = 124, destination = $transp_3$
- offset `0x8dd` : taille = 144, destination = $transp_3$

Nous pouvons alors remarquer que les `Transputer 1`, `Transputer 2` et `Transputer 3` font exactement la même chose que `Transputer 0` : une fois qu'ils ont reçu leur propre code de démarrage, ils envoient à leur tour un boot code à leurs enfants. Il est donc possible de découper un peu plus finement la liste précédente :
Envoi du boot code des transputers 1, 2 et 3 :

- offset `0xf9` : taille = 113, destination = $transp_1$
- offset `0x176` : taille = 113, destination = $transp_2$
- offset `0x1f3` : taille = 113, destination = $transp_3$

Envoi du boot code des enfants des transputers 1, 2, 3 :

- offset `0x270` : taille = 49, destination = $transp_1$ / $transp_4$
- offset `0x2ad` : taille = 49, destination = $transp_1$ / $transp_5$
- offset `0x2ea` : taille = 49, destination = $transp_1$ / $transp_6$

- offset `0x327` : taille = 49, destination = $transp_2$ / $transp_7$
- offset `0x364` : taille = 49, destination = $transp_2$ / $transp_8$
- offset `0x3a1` : taille = 49, destination = $transp_2$ / $transp_9$

- offset `0x3de` : taille = 49, destination = $transp_3$ / $transp_{10}$
- offset `0x41b` : taille = 49, destination = $transp_3$ / $transp_{11}$
- offset `0x458` : taille = 49, destination = $transp_3$ / $transp_{12}$

Envoi du "code utile" à chaque transputer :

- offset `0x495` : taille = 92, destination = $transp_1$ / $transp_4$
- offset `0x4fd` : taille = 92, destination = $transp_1$ / $transp_5$
- offset `0x565` : taille = 152, destination = $transp_1$ / $transp_6$
- offset `0x609` : taille = 112, destination = $transp_2$ / $transp_7$
- offset `0x685` : taille = 168, destination = $transp_2$ / $transp_8$
- offset `0x739` : taille = 96, destination = $transp_2$ / $transp_9$
- offset `0x7a5` : taille = 164, destination = $transp_3$ / $transp_{10}$
- offset `0x855` : taille = 124, destination = $transp_3$ / $transp_{11}$
- offset `0x8dd` : taille = 144, destination = $transp_3$ / $transp_{12}$

La suite de l'analyse va donc se concentrer sur les charges utiles de chaque transputer (offsets `0x495` et suivants).

5.3 Décompilation

L'étape suivante consiste à décompiler le code utile de chaque transputer. Il s'agit d'une tâche assez fastidieuse, mais relativement simple. Les transputers étant des machines à pile, il est très simple de remplacer chaque instruction en un équivalent C, puis de regrouper les résultats en quelque chose de plus lisible. Je ne donne ici que le détail du premier, tous les autres ont subi le même traitement.

5.3.1 Exemple du Transputer 4

Le code original est le suivant :

```
========== B E G I N N I N G   O F   P R O C E D U R E ==========


            transp4_input:
800004b9        ldl                 3
800004ba        ldl                 2
800004bb        ldl                 4
800004bc        in
800004bd        ret
                        ; endp

========== B E G I N N I N G   O F   P R O C E D U R E ==========


            transp4_output:
800004bf        ldl                 3
800004c0        ldl                 2
800004c1        ldl                 4
800004c2        out
800004c3        ret
                        ; endp
========== B E G I N N I N G   O F   P R O C E D U R E ==========


            transp4_process:
800004c5        ajw                 -5
800004c7        ldc                 0
800004c8        stl                 1
800004c9        ldc                 0
800004ca        ldlp                1
800004cb        sb

800004cd        ldc                 12
800004ce        stl                 0
800004cf        ldlp                2
800004d0        mint
800004d2        ldnlp               4
800004d3        ldl                 6
800004d4        call                transp4_input

800004d6        ldc                 0
800004d7        stl                 0
800004d8        ldl                 0
800004d9        ldlp                2
800004da        bsub
800004db        lb
800004dc        ldlp                1
800004dd        lb
800004de        bsub
800004df        ldc                 0xff
800004e1        and
800004e3        ldlp                1
800004e4        sb
800004e6        ldl                 0
800004e7        adc                 1
800004e8        stl                 0
800004e9        ldc                 12
800004ea        ldl                 0
800004eb        gt
800004ec        cj                  0x800004ef

800004ed        j                   0x800004d8
```

```
800004ef          ldc                 1
800004f0          stl                 0
800004f1          ldlp                1
800004f2          mint
800004f4          ldl                 6
800004f5          call                transp4_output
800004f7          j                   0x800004cd
                          ; endp
```

Il y a 3 procédures par transputer. Les deux premières sont tout simplement des procédures de lecture, et d'écriture sur les liens inter-transputers. Nous nous concentrons donc sur la méthode à l'adresse `0x800004c5`. La première instruction est ignorée, il ne s'agit que de déclarer la zone des variables locales. Ensuite nous transformons le code :

```
   ldc 0
   stl 1  --->     locale_1 = 0
```

```
   ldc  0
   ldlp 1
   sb       --->  *(uint8_t *)&locale1 = 0
```

```
   ldc 12
   stl 0  --->     locale0 = 12
```

```
   ldc     12
   stl     0
   ldlp    2
   mint
   ldnlp   4
   ldl     6
   call transp4_input  ---> lecture de 12 octets a &locale_2
```

```
   ldc    0
   stl    0     ---> locale0 = 0

loop:
   ldl    0     ---> locale0
   ldlp   2     ---> &locale2
   bsub         ---> &locale2[locale0]
   lb           ---> locale2[locale0]

   ldlp   1     ---> &locale1
   lb           ---> locale1

   bsub         ---> locale2[locale0] + locale1

   ldc    0xff  ---> 0xFF
   and          ---> (locale2[locale0] + locale1) & 0xFF

   ldlp   1
   sb           ---> locale1 = (locale2[locale0] + locale1) & 0xFF

   ldl    0     ---> locale0
   adc    1     ---> locale0 + 1
   stl    0     ---> locale0 = locale0 + 1
```

```
ldc    12    ----> 12
ldl    0     ----> locale0
gt           ----> 12 > locale0
cj     exit  ----> if (locale0 >= 12) goto exit
j      loop  ----> if (locale0 < 12) goto loop
```

Petit à petit, nous obtenons le code C :

```
void transp4(uint32_t *input, uint32_t *output) {
    static uint8_t sum = 0;
    uint8_t *key = (uint8_t *)input;
    for (int i=0; i<12; i++) {
        sum += *key++;
    }
    *output = sum;
}
```

Il est possible de faire de même avec tous les transputers. Le code final, simulant l'intégralité du réseau, se trouve en annexe E. La seule petite difficulté concerne les `Transputers 11` et `Transputers 12`, car ils communiquent entre eux, en plus de communiquer avec leurs parents. Heureusement, les communications sont synchrones, et il suffit d'entremêler le code des deux transputers en une seule entité.

5.4 Déchiffrement des données

En regardant de plus près le code du simulateur, on aperçoit que le système de chiffrement souffre (heureusement) d'une faiblesse : les 12 premiers octets sont chiffrés avec un XOR d'une transformation simple de la clé.

Lors du désassemblage, une chaîne de caractère attire mon attention : le résultat du déchiffrement semble être un fichier compressé en BZIP2. Nous pouvons exploiter cette information pour déduire quels sont les premiers octets du flux déchiffré.

L'en-tête d'un fichier BZIP2 démarre avec une signature 10 octets :

```
uint8_t bz_header[] = {'B', 'Z', 'h', '9', 0x31, 0x41, 0x59, 0x26, 0x53,
    0x59};
```

À partir de ces 10 octets, il est possible de deviner une partie des 10 premiers caractères de la clé. En effet, seuls 7 bits sur les 8 bits peuvent être calculés. Sur les

$$12 \times 8 = 96$$

bits de la clé, nous en connaissons donc, de manière sûre, 70.

$$\begin{aligned} key[0] &= \lfloor \frac{(bz\_header[0] \oplus ciphered[0])}{2} \rfloor \pm \texttt{0x80} \\ key[1] &= \lfloor \frac{(bz\_header[1] \oplus ciphered[1]) - 1}{2} \rfloor \pm \texttt{0x80} \\ key[2] &= \lfloor \frac{(bz\_header[2] \oplus ciphered[2]) - 2}{2} \rfloor \pm \texttt{0x80} \end{aligned}$$

$$\begin{aligned} & \dots \\ key[10] \quad &= \quad \lfloor \frac{((bz\_header[10] \oplus ciphered[10]) - 10}{2} \rfloor \pm \texttt{0x80} \end{aligned}$$

Il n'y a plus qu'a tenter de trouver les $2 \times 8 + 10 = 26$ bits manquants. Pour cela, il suffit d'itérer sur toutes les valeurs possibles, puis de tenter de déchiffrer le début du fichiers. Il apparaît que dans la majorité des fichiers BZIP2, l'octet à l'offset `0x0E` à son bit de poids fort à 0, et les octets aux offsets `0x12` et `0x13` sont souvent égaux à `0xFF`. En utilisant cette heuristique simple, il est possible d'accélérer la recherche de la clé, et de ne tenter de déchiffrer totalement qu'avec un sous-ensemble des clés générées.

```
void search_key() {
    FILE *f = fopen("stage5_encrypted", "rb");
    unsigned int fileSize = 250606;
    uint8_t *data = (uint8_t *) malloc(fileSize);
    uint8_t *o_data = (uint8_t *) malloc(fileSize);
    fread(o_data, fileSize, 1, f);
    fclose(f);

    int cnt = 0;
    for (int i_f=0; i_f<65536; i_f++) {
        uint8_t bz_header[] = {'B', 'Z', 'h', '9', 0x31, 0x41, 0x59, 0
            x26, 0x53, 0x59, 0, 0};
        bz_header[10] = i_f & 255;
        bz_header[11] = (i_f >> 8) & 255;

        for (int i_v=0; i_v<4096; i_v++) {
            uint8_t data_key[12];
            bool valid = true;
            for (int i=0; i<12; i++) {
                // i + 2*k[i] = bz_header[i] ^ data[i]
                // k[i] = (bz_header[i] ^ data[i] - i) / 2
                uint8_t mask = (bz_header[i] ^ o_data[i]);
                mask -= (uint8_t) i;
                if (mask & 1) valid = false;
                uint8_t b_iv = (((i_v >> i)) & 1) << 7;
                data_key[i] = b_iv | (mask / 2);
            }
            if (!valid) continue;

            memcpy(data, o_data, 64);
            uncipher(data_key, data, 64);
            if ((data[0x0E] & 0x80) == 0
              && data[0x12] == 0xFF && data[0x13] == 0xFF) {
                printf("%d (%d): ", i_f, ++cnt);
                for (int k=0; k<12; k++) printf("%02X", (uint8_t)
                    data_key[k]);

                memcpy(data, o_data, fileSize);
                uncipher(data_key, data, fileSize);

                uint8_t digest[CC_SHA256_DIGEST_LENGTH];
                CC_SHA256(data, fileSize, digest);
                printf(" ");
                for (int i=0; i<32; i++) printf("%02X", digest[i]);
                puts("");

                uint8_t expected[] = {
                    0x91, 0x28, 0x13, 0x51, 0x29, 0xd2, 0xbe, 0x65,
                    0x28, 0x09, 0xf5, 0xa1, 0xd3, 0x37, 0x21, 0x1a,
                    0xff, 0xad, 0x91, 0xed, 0x58, 0x27, 0x47, 0x4b,
                    0xf9, 0xbd, 0x7e, 0x28, 0x5e, 0xce, 0xf3, 0x21
                };

                if (memcmp(expected,
                           digest,
                           CC_SHA256_DIGEST_LENGTH) == 0) {
                    char name[256];
                    sprintf(name, "final-%02X%02X.tar.bz2", bz_header
```

```
                    [10], bz_header[11]);
                FILE *f = fopen(name, "wb");
                fwrite(data, fileSize, 1, f);
                fclose(f);
                exit(0);
            }
        }
    }
}
```

Après 5 à 10 minutes de suspens, nous obtenons la clé suivante :

`5ED49B7156FCE47DE976DAC5`.

Chapitre 6

Stage 6 - Les images

L'archive obtenue est décompressée. Elle contient une image :

FIGURE 6.1 – Première image

6.1 Premier effort...

Le fichier est au format JPG. Ce type de fichier débute par le couple d'octets `0xFF 0xF8`, et se termine par le couple `0xFF 0xF9`. Dans le cas présent, la signature de fin n'est pas présente, ce qui laisse à penser qu'un autre fichier à été ajouté à la suite de l'image JPG originale. En recherchant les signatures `0xFF 0xF9`, on trouve assez rapidement qu'un fichier BZIP2 a été ajouté à partir de l'offset `0xD7D0`. Il suffit de l'extraire pour passer à l'étape suivante.

6.2 Deuxième effort...

Le fichier BZIP2 extrait contient une image PNG.

FIGURE 6.2 – Deuxième image

En analysant le fichier avec la commande `pngcheck`, du la bibliothèque libPNG [1], nous obtenons une erreur :

```
$ pngcheck congratulations.png
congratulations.png  illegal reserved-bit-set chunk sTic
ERROR: congratulations.png
```

Il semblerait que des chunks `sTic` aient été ajoutés dans le fichier. Ces chunks n'ont pas de signification pour le décodeur PNG, et sont tout simplement ignorés, d'où le fait que l'image soit affichée correctement.

Grâce à un petit programme en C, nous pouvons faire la liste de tous ces chunks `sTic`, les concaténer, pour obtenir de nouvelles données :

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

int main (int argc, char const *argv[])
{
    FILE *f = fopen("congratulations.png", "rb");
    fseek(f, 0x5b, SEEK_SET);
    FILE *d = fopen("sTic.zlib", "wb");
    int total = 0;
    while (1) {
        uint32_t length, magic, crc;
        fread(&length, 4, 1, f);
        fread(&magic, 4, 1, f);
        printf("%x\n", magic);
        if (magic != 'ciTs') {
            break;
        }
        length = htonl(length);
        printf("CHUNK length 0x%x, at 0x%x\n", length, ftell(f));

        void *b = malloc(length);
        fread(b, length, 1, f);
        fwrite(b, length, 1, d);
```

1. `http://www.libpng.org/pub/png/apps/pngcheck.html`

```
            free(b);
            total += length;

            fread(&crc, 4, 1, f);
        }
        return 0;
}
```

6.3 Troisième effort...

Le fichier extrait du PNG est un fichier compressé avec la ZLIB. Nous le décompressons simplement avec la commande :

```
$ cat sTic.zlib | perl -MCompress::Zlib -e '\''undef $/; print
    uncompress(<>)'\ > out
```

Le résultat est au format BZIP2

```
$ file out
out: bzip2 compressed data, block size = 900k
```

Une fois décompressé, nous obtenons une nouvelle image. Il s'agit cette fois-ci d'une image TIFF :

FIGURE 6.3 – Troisième image

Cette fois-ci, je décide de lire le contenu de ce fichier grâce à la bibliothèque Python PIL.

```
>> from PIL import Image
>> img = list(Image.open("congratulations.tiff").getdata())
>> img[:100]
[(0, 1, 0), (0, 0, 0), (0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 1, 0), (1,
    0, 0), (1, 0, 0), (0, 1, 0), (1, 0, 0), (1, 0, 0), (0, 0, 0), (0,
```

```
0, 0), (1, 1, 0), (1, 0, 0), (0, 1, 0), (0, 0, 0), (1, 1, 0), (0,
0, 0), (0, 1, 0), (0, 1, 0), (0, 0, 0), (0, 0, 0), (0, 1, 0), (0,
1, 0), (0, 1, 0), (1, 0, 0), (0, 1, 0), (0, 0, 0), (1, 0, 0), (0,
1, 0), (1, 0, 0), (0, 1, 0), (0, 1, 0), (0, 0, 0), (1, 1, 0), (0,
1, 0), (0, 1, 0), (1, 0, 0), (0, 1, 0), (0, 1, 0), (0, 0, 0), (0,
0, 0), (1, 0, 0), (0, 0, 0), (0, 0, 0), (0, 0, 0), (1, 0, 0), (0,
0, 0), (1, 0, 0), (0, 1, 0), (0, 1, 0), (1, 1, 0), (0, 0, 0), (1,
0, 0), (0, 0, 0), (0, 0, 0), (0, 0, 0), (0, 0, 0), (0, 0, 0), (0,
0, 0), (0, 1, 0), (1, 0, 0), (0, 1, 0), (1, 0, 0), (0, 0, 0), (0,
0, 0), (0, 1, 0), (1, 1, 0), (1, 1, 0), (1, 1, 0), (1, 1, 0), (1,
1, 0), (1, 1, 0), (1, 1, 0), (1, 1, 0), (1, 1, 0), (1, 1, 0), (1,
1, 0), (1, 1, 0), (1, 1, 0), (1, 1, 0), (1, 1, 0), (1, 1, 0), (1,
1, 0), (1, 1, 0), (1, 1, 0), (1, 1, 0), (1, 1, 0), (1, 1, 0), (1,
1, 0), (1, 1, 0), (1, 1, 0), (1, 1, 0), (1, 1, 0), (1, 1, 0), (1,
1, 0), (1, 1, 0), (1, 1, 0), (1, 1, 0)]
```

Visiblement, les bits de poids faible des canaux rouge et vert sont utilisés pour transporter de l'information. Il est possible de l'extraire grâce à ce petit programme C :

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

int main (int argc, char const *argv[])
{
    FILE *f = fopen("congratulations.tiff", "rb");
    uint8_t *buffer = (uint8_t *) malloc(904520);
    fread(buffer, 904520, 1, f);

    FILE *d = fopen("bz_in_tiff.bz2", "wb");

    int idx = 0;
    uint8_t c = 0, b;
    uint8_t *ptr = buffer + 0x80;
    for (int i=0; i<300000; i++) {
        b = ((ptr[0] & 1) << 1) | (ptr[1] & 1);
        c <<= 2;
        c |= b;
        idx+=2;
        if (idx == 8) {
            fwrite(&c, 1, 1, d);
            idx = 0;
            c = 0;
        }
        ptr += 3;
    }
    return 0;
}
```

6.4 Quatrième effort

Le fichier obtenu est un fichier BZIP2, qui contient... encore une image.

FIGURE 6.4 – Quatrième image

Cette image est au format GIF. L'image contient une palette de couleurs, et les pixels sont des index dans cette palette. Afin de passer cette étape, il suffit de jouer avec la palette de couleur dans un programme comme The Gimp. En modifiant la couleur à l'index 2, nous voyons apparaître ceci :

FIGURE 6.5 – Victoire !

Et voilà, c'est terminé, l'adresse qu'il fallait trouver est :

`1713e7c1d0b750ccd4e002bb957aa799@challenge.sstic.org`

Annexe A

Décodage Duckyscript

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

char byteToChar(unsigned char b) {
    if (b >= 4 && b < 30) return b - 4 + 'a';
    if (b >= 30 && b <= 39) {
        const char *map = "1234567890";
        return map[b - 30];
    }
    switch (b) {
        case 0x1e: return '!'; // 1
        case 0x1f: return '@'; // 2
        case 0x20: return '#'; // 3
        case 0x21: return '$'; // 4
        case 0x22: return '%'; // 5
        case 0x23: return '^'; // 6
        case 0x24: return '&'; // 7
        case 0x25: return '*'; // 8
        case 0x26: return '('; // 9
        case 0x27: return ')'; // 0
        case 0x2C: return ' ';
        case 0x2D: return '-'; // -
        case 0x2E: return '='; // +
        case 0x2F: return '['; // {
        case 0x30: return ']'; // }
        case 0x31: return '\\';
        case 0x33: return ':';
        case 0x34: return '\'';
        case 0x35: return '`';
        case 0x36: return ',';
        case 0x37: return '.';
        case 0x38: return '/';
    }

    printf("cannot convert byte %d\n", b);
    exit(1);
    return 0;
}

char shiftChar(char c) {
    if (c >= 'a' && c <= 'z') return c + 'A' - 'a';

    if (c == '1') return '!';
    if (c == '2') return '@';
    if (c == '3') return '#';
    if (c == '4') return '$';
    if (c == '5') return '%';
    if (c == '6') return '^';
    if (c == '7') return '&';
    if (c == '8') return '*';
    if (c == '9') return '(';
```

```
    if (c == '0') return ')';
    if (c == '-') return '_';
    if (c == '=') return '+';
    if (c == '[') return '{';
    if (c == ']') return '}';
    if (c == '\\') return '|';
    if (c == ';') return ':';
    if (c == '\'') return '"';
    if (c == '`') return '~';
    if (c == ',') return '<';
    if (c == '.') return '>';
    if (c == '/') return '?';

    printf("CANNOT SHIFT %c\n", c);
    exit(1);
}

const char *byteToFunctionKey(unsigned char b) {
    switch (b) {
        case 0x3A: return "F1";
        case 0x3B: return "F2";
        case 0x3C: return "F3";
        case 0x3D: return "F4";
        case 0x3E: return "F5";
        case 0x3F: return "F6";
        case 0x40: return "F7";
        case 0x41: return "F8";
        case 0x42: return "F9";
        case 0x43: return "F10";
        case 0x44: return "F11";
        case 0x45: return "F12";
        case 0x99: return "F??";
    }
    printf("bad function key %d\n", b);
    exit(1);
}

int main (int argc, char const *argv[])
{
    FILE *f = fopen("inject.bin", "rb");
    size_t l = 34253730;

    unsigned char *c = (unsigned char *) malloc(l);
    unsigned char *start = c;
    fread(c, l, 1, f);
    fclose(f);

    unsigned char *e = c + l;
    int last_was_char = 0;

#define FLUSH if (last_was_char) { puts(""); last_was_char = 0; }

    while (c < e) {
        unsigned char first = *c++;
        unsigned char second = *c++;

        if (first == 0) {
            FLUSH;
            printf("DELAY %d\n", second);
        }
        else if (second == 0x00) {
                if (first == 0x28) {FLUSH; printf("ENTER\n"); }
            else if (first == 0x29) {FLUSH; printf("ESCAPE\n"); }
            else if (first == 0x2B) {FLUSH; printf("TAB\n"); }
            else if (first == 0x2C) {FLUSH; printf("SPACE\n"); }
            else if (first == 0x39) {FLUSH; printf("CAPSLOCK\n"); }
            else if (first == 0x46) {FLUSH; printf("PRINTSCREEN\n"); }
            else if (first == 0x47) {FLUSH; printf("SCROLLLOCK\n"); }
            else if (first == 0x48) {FLUSH; printf("BREAK\n"); }
            else if (first == 0x4A) {FLUSH; printf("HOME\n"); }
            else if (first == 0x4B) {FLUSH; printf("PAGEUP\n"); }
            else if (first == 0x4C) {FLUSH; printf("DELETE\n"); }
            else if (first == 0x4D) {FLUSH; printf("END\n"); }
            else if (first == 0x4E) {FLUSH; printf("PAGEDOWN\n"); }
```

```
            else if (first == 0x4F) {FLUSH; printf("RIGHTARROW\n"); }
            else if (first == 0x50) {FLUSH; printf("LEFTARROW\n"); }
            else if (first == 0x51) {FLUSH; printf("DOWNARROW\n"); }
            else if (first == 0x52) {FLUSH; printf("UPARROW\n"); }
            else if (first == 0x53) {FLUSH; printf("NUMLOCK\n"); }
            else if (first == 0x65) {FLUSH; printf("MENU\n"); }
            else if (first == 0x81) {FLUSH; printf("SYSTEMPOWER\n"); }
            else if (first == 0x82) {FLUSH; printf("SYSTEMSLEEP\n"); }
            else if (first == 0x83) {FLUSH; printf("SYSTEMWAKE\n"); }
            else if (first == 0xB5) {FLUSH; printf("STOP\n"); }
            else if (first == 0xCD) {FLUSH; printf("PLAY\n"); }
            else if (first == 0xE1) {FLUSH; printf("SHIFT\n"); }
            else if (first == 0xE2) {FLUSH; printf("MUTE\n"); }
            else if (first == 0xE3) {FLUSH; printf("WINDOWS\n"); }
            else if (first == 0xE9) {FLUSH; printf("VOLUMEUP\n"); }
            else if (first == 0xEA) {FLUSH; printf("VOLUMEDOWN\n"); }
            else {
                if (!last_was_char) {
                    // printf("REM offset = %ld\n", c - 2 - start);
                    printf("STRING ");
                }
                printf("%c", byteToChar(first));
                last_was_char = 1;
            }
        }
        else if (second == 0x02) {
            if (!last_was_char) {
                // printf("REM offset = %ld\n", c - 2 - start);
                printf("STRING ");
            }
            printf("%c", shiftChar(byteToChar(first)));
            last_was_char = 1;
        }
        else if (second == 0x08) {
            FLUSH;
            printf("WINDOWS %c\n", byteToChar(first));
        }
        else if (second == 0x01) {
            FLUSH;
            if (first == 0x29) printf("CONTROL ESCAPE\n");
            else if (first == 0x48) printf("CONTROL PAUSE\n");
            else if (first >= 0x3A && first <= 0x45) printf("CONTROL F%d
                \n", first - 0x39);
            else printf("CONTROL %c\n", byteToChar(first));
        }
        else if (second == 0xE1) {
            FLUSH;
                if (first == 0x2B) printf("SHIFT TAB\n");
            else if (first == 0x49) printf("SHIFT INSERT\n");
            else if (first == 0x4A) printf("SHIFT HOME\n");
            else if (first == 0x4B) printf("SHIFT PAGEUP\n");
            else if (first == 0x4C) printf("SHIFT DELETE\n");
            else if (first == 0x4D) printf("SHIFT END\n");
            else if (first == 0x4E) printf("SHIFT PAGEDOWN\n");
            else if (first == 0x4F) printf("SHIFT RIGHTARROW\n");
            else if (first == 0x50) printf("SHIFT LEFTARROW\n");
            else if (first == 0x51) printf("SHIFT DOWNARROW\n");
            else if (first == 0x52) printf("SHIFT UPARROW\n");
            else if (first == 0xE3) printf("SHIFT WINDOWS\n");
        }
        else if (second == 0xE2) {
            FLUSH;
            if (first == 0x29) printf("ALT ESCAPE\n");
            else if (first == 0x2C) printf("ALT SPACE\n");
            else if (first == 0x2B) printf("ALT TAB\n");
            else if (first >= 0x3A && first <= 0x45) printf("ALT F%d\n",
                first - 0x39);
            else printf("ALT %c\n", byteToChar(first));
        }
        else {
            printf("unknown %x %x\n", first, second);
        }
}
```

```
    return 0;
}
```

Annexe B

Reconstruction PowerShell

```
import sys
import base64

begin_mark = "write_file_bytes([Convert]::FromBase64String('"
end_mark = "'));}else{write_file_bytes([Convert]::FromBase64String('
    VAByAHkASABhAHIAZABlAHIA'));}"

zip = ""

with open("decoded.txt") as f:
    for line in f:
        if line[:8] == "STRING Z":
            content = line[7:]
            unicode_data = base64.b64decode(content)
            data = ""
            for i in xrange(0, len(unicode_data), 2):
                data += unicode_data[i]
            begin = data.find(begin_mark)
            end = data.find(end_mark)
            if begin != -1 and end != -1:
                begin += len(begin_mark)
                inner = data[begin : end]
                zip += base64.b64decode(inner)

f = open('stage2.zip', 'wb')
f.write(zip)
```

Annexe C

Dessin de la trace USB

```
#import "Painter.h"

@implementation Painter

union usb_packet {
    uint32_t value;
    struct {
        unsigned left_btn:1;
        unsigned right_btn:1;
        unsigned middle_btn:1;
        unsigned one:1;
        signed x_sign:1;
        signed y_sign:1;
        unsigned x_overflow:1;
        unsigned y_overflow:1;
        unsigned delta_x:8;
        unsigned delta_y:8;
        unsigned zero:8;
    };
};

- (void)drawRect:(NSRect)dirtyRect {
    FILE *f = fopen("paint.cap", "rb");
    size_t size = 2347070;

    uint8_t *data = (uint8_t *) malloc(size);
    fread(data, size, 1, f);

    uint8_t *ptr = data + 0x23e;
    int it = 1;

    int x = 200, y = 200;

    while (ptr < data + size) {
        if (it) {
            uint32_t *payload = (uint32_t *) (ptr + 64);
            union usb_packet packet;
            packet.value = *payload;
            int dx = packet.delta_x;
            int dy = packet.delta_y;
            if (dx & 0x80) dx |= (-1)<<8;
            if (dy & 0x80) dy |= (-1)<<8;

            if (packet.left_btn) {
                NSRectFill(NSMakeRect(x, self.bounds.size.height - y, 1,
                    1));
            }

            x += dx;
            y += dy;
```

```
        }
        ptr += (it ? 84 : 80);
        it = 1 - it;
    }


    fclose(f);
    free(data);
}
```

```
@end
```

Annexe D

Déchiffrement Serpent-1

```
#include <iostream>
#include <cryptopp/cryptlib.h>
#include <cryptopp/modes.h>
#include <cryptopp/files.h>
#include <cryptopp/hex.h>
#include <cryptopp/serpent.h>
#include <string.h>

using namespace std;
using namespace CryptoPP;

int main (int argc, char const *argv[]) {
    uint8_t key[] = {
        0x66, 0xc1, 0xba, 0x5e, 0x8c, 0xa2, 0x9a, 0x8a,
        0xb6, 0xc1, 0x05, 0xa9, 0xbe, 0x9e, 0x75, 0xfe,
        0x0b, 0xa0, 0x79, 0x97, 0xa8, 0x39, 0xff, 0xea,
        0xe9, 0x70, 0x0b, 0x00, 0xb7, 0x26, 0x9c, 0x8d
    };

    byte iv[Serpent::BLOCKSIZE + 1];
    strcpy((char *)iv, "SSTIC2015-Stage3");

    try {
        CBC_CTS_Mode< Serpent >::Decryption d;
        d.SetKeyWithIV(key, sizeof(key), iv);

        FileSource input("encrypted", true,
            new StreamTransformationFilter(d,
                new FileSink("stage3-unciphered")
            )
        );
    }
    catch(const CryptoPP::Exception& e) {
        cerr << e.what() << endl;
    }

    return 0;
}
```

Annexe E

Simulateur du réseau de ST20

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

#include <CommonCrypto/CommonCrypto.h>
#include <dispatch/dispatch.h>

uint8_t transp1(int gi, uint8_t *input);
uint8_t transp2(int gi, uint8_t *input);
uint8_t transp3(int gi, uint8_t *input);
uint8_t transp4(int gi, uint8_t *input);
uint8_t transp5(int gi, uint8_t *input);
uint8_t transp6(int gi, uint8_t *input);
uint8_t transp7(int gi, uint8_t *input);
uint8_t transp8(int gi, uint8_t *input);
uint8_t transp9(int gi, uint8_t *input);
uint8_t transp10(int gi, uint8_t *input);
uint8_t transp11_12(int gi, uint8_t *input);

#define MEM_SIZE        4096
#define LOCAL_OFFSET    400

void uncipher(const uint8_t *key, uint8_t *ciphered, size_t length) {
    uint8_t m_key[12];
    memcpy(m_key, key, 12);

    for (int i=0; i<length; i++) {
        int l4 = i % 12;
        uint8_t t1 = transp1(i, (uint8_t *)m_key);
        uint8_t t2 = transp2(i, (uint8_t *)m_key);
        uint8_t t3 = transp3(i, (uint8_t *)m_key);
        uint8_t t = t1 ^ t2 ^ t3;
        uint8_t m = (l4 + 2 * m_key[l4]);
        ciphered[i] ^= m;
        m_key[l4] = t;
        l4 = (l4 + 1) % 12;
    }
}

int main (int argc, char const *argv[])
{
    uint8_t ciphered[] = {0x1d, 0x87, 0xc4, 0xc4, 0xe0, 0xee, 0x40, 0x38
        , 0x3c, 0x59, 0x44, 0x7f, 0x23, 0x79, 0x8d, 0x9f, 0xef, 0xe7, 0
        x4f, 0xb8, 0x24, 0x80, 0x76, 0x6e};

    uncipher((const uint8_t *)"*SSTIC-2015*", ciphered, sizeof(ciphered)
        );
    for (int i=0; i<sizeof(ciphered); i++) printf("%c", ciphered[i]);
    printf("\n");
```

```
    return 0;
}

uint8_t transp1(int gi, uint8_t *input) {
    return transp4(gi, input) ^ transp5(gi, input) ^ transp6(gi, input);
}

uint8_t transp2(int gi, uint8_t *input) {
    return transp7(gi, input) ^ transp8(gi, input) ^ transp9(gi, input);
}

uint8_t transp3(int gi, uint8_t *input) {
    return transp10(gi, input) ^ transp11_12(gi, input);
}

uint8_t transp4(int gi, uint8_t *input) {
    static uint8_t sum;
    if (gi == 0) sum = 0;
    uint8_t *key = (uint8_t *)input;
    for (int i=0; i<12; i++) {
        sum += *key++;
    }
    return sum;
}

uint8_t transp5(int gi, uint8_t *input) {
    static uint8_t sum;
    if (gi == 0) sum = 0;
    uint8_t *key = (uint8_t *)input;
    for (int i=0; i<12; i++) {
        sum ^= *key++;
    }
    return sum;
}

uint8_t transp6(int gi, uint8_t *input) {
    static uint8_t memory[MEM_SIZE];
    static uint32_t *l = (uint32_t *) (memory + LOCAL_OFFSET);

    if (gi == 0) {
        l[2] = 0;
        l[1] = 0;
        l[3] = 0;
    }

    l[0] = 12;
    memcpy(memory + LOCAL_OFFSET + 16, input, 12);

    if (l[3] == 0) {
        for (l[0] = 0; l[0] < 12; l[0]++) {
            l[1] = (memory[LOCAL_OFFSET + 16 + l[0]] + l[1]) & 0xFFFF;
        }
        l[3] = 1;
    }

    l[1] = (((((l[1] & 0x8000) >> 15) ^ ((l[1] & 0x4000) >> 14)) & 0
        xFFFF) ^ ((l[1] << 1) & 0xFFFF)) & 0xFFFF;

    memory[LOCAL_OFFSET + 8] = l[1] & 0xFF;

    l[0] = 1;
    return l[2];
}

uint8_t transp7(int gi, uint8_t *input) {
    static uint8_t memory[MEM_SIZE];
    static uint32_t *l = (uint32_t *) (memory + LOCAL_OFFSET);

    if (gi == 0) {
        l[3] = 0;
    }

    l[0] = 12;
    memcpy(memory + LOCAL_OFFSET + 16, input, 12);
```

```
    l[1] = 0;
    l[2] = 0;

    for (l[0] = 0; l[0] < 6; l[0]++) {
        l[1] = (memory[LOCAL_OFFSET + 16 + l[0]] + l[1]) & 0xFF;
        l[2] = (memory[LOCAL_OFFSET + 16 + l[0] + 6] + l[2]) & 0xFF;
    }

    memory[LOCAL_OFFSET + 12] = (l[2] ^ l[1]) & 0xFF;

    l[0] = 1;
    return l[3];
}

uint8_t transp8(int gi, uint8_t *input) {
    static uint8_t memory[MEM_SIZE];
    static uint32_t *l = (uint32_t *) (memory + LOCAL_OFFSET);

    if (gi == 0) {
        l[3] = 0;
        l[4] = 0;

        for (l[2] = 0; l[2] < 4; l[2]++) {
            for (l[0] = 0; l[0] < 12; l[0]++) {
                memory[LOCAL_OFFSET + 20 + l[2] * 12 + l[0]] = 0;
            }
        }
    }

    l[0] = 12;
    memcpy(memory + LOCAL_OFFSET + 20 + l[4] * 12, input, 12);

    l[4] = (l[4] + 1) % 4;

    memory[LOCAL_OFFSET + 12] = 0;

    for (l[2] = 0; l[2] < 4; l[2]++) {
        l[1] = 0;

        for (l[0] = 0; l[0] < 12; l[0]++) {
            l[1] = (memory[LOCAL_OFFSET + 20 + l[2] * 12 + l[0]] + l[1])
                & 0xFF;
        }

        memory[LOCAL_OFFSET + 12] = (l[3] ^ l[1]) & 0xFF;
    }

    l[0] = 1;
    return l[3];
}

uint8_t transp9(int gi, uint8_t *input) {
    static uint8_t memory[MEM_SIZE];
    static uint32_t *l = (uint32_t *) (memory + LOCAL_OFFSET);

    if (gi == 0) {
        l[1] = 0;
    }

    l[0] = 12;
    memcpy(memory + LOCAL_OFFSET + 8, input, 12);

    memory[LOCAL_OFFSET + 4] = 0;

    for (l[0] = 0; l[0] < 12; l[0]++) {
        memory[LOCAL_OFFSET + 4] = ((memory[LOCAL_OFFSET + 8 + l[0]] <<
            (l[0] & 7)) ^ l[1]) & 0xFF;
    }

    l[0] = 1;
    return l[1];
}
```

```
uint8_t transp10(int gi, uint8_t *input) {
    static uint8_t memory[MEM_SIZE];
    static uint32_t *l = (uint32_t *) (memory + LOCAL_OFFSET);

    if (gi == 0) {
        l[3] = 0;
        l[2] = 0;

        for (l[0] = 0; l[0] < 4; l[0]++) {
            for (l[1] = 0; l[1]<12; l[1]++) {
                memory[LOCAL_OFFSET + 16 + l[0] * 12 + l[1]] = 0;
            }
        }
    }

    l[0] = 12;
    memcpy(memory + LOCAL_OFFSET + 16 + l[2] * 12, input, 12);
    l[2] = (l[2] + 1) % 4;

    l[1] = 0;
    for (l[0] = 0; l[0] < 4; l[0]++) {
        l[1] = (memory[LOCAL_OFFSET + 16 + l[0] * 12] + l[1]) & 0xFF;
    }

    memory[LOCAL_OFFSET + 12] = memory[LOCAL_OFFSET + 16 + (l[1] & 3) *
        12 + (((l[1] >> 4) % 12) & 0xFF)];

    l[0] = 1;
    return l[3];
}

uint8_t transp11_12(int gi, uint8_t *input) {
    static uint8_t memory_11[MEM_SIZE], memory_12[MEM_SIZE];
    static uint32_t *l_11 = (uint32_t *) (memory_11 + LOCAL_OFFSET);
    static uint32_t *l_12 = (uint32_t *) (memory_12 + LOCAL_OFFSET);

    if (gi == 0) {
        l_11[1] = 0;
        l_11[2] = 0;

        l_12[2] = 0;
        l_12[1] = 0;

        for (l_12[0] = 0; l_12[0] < 12; l_12[0]++) {
            memory_12[LOCAL_OFFSET + 12 + l_12[0]] = 0;
        }
    }

    l_11[0] = 12;
    memcpy(memory_11 + LOCAL_OFFSET + 12, input, 12);

    memory_11[LOCAL_OFFSET + 4] = 0;
    memory_11[LOCAL_OFFSET + 4] = (memory_11[LOCAL_OFFSET + 12] ^
        memory_11[LOCAL_OFFSET + 12 + 3] ^ memory_11[LOCAL_OFFSET + 12
        + 7]) & 0xFF;

    l_11[0] = 1;
    // envoi de l[1] au frere

    uint8_t input_brother = l_11[1];

    /* trans12 */

    memory_12[LOCAL_OFFSET + 4] = 0;
    memory_12[LOCAL_OFFSET + 4] = (memory_12[LOCAL_OFFSET + 12 + 1] ^
        memory_12[LOCAL_OFFSET + 12 + 5] ^ memory_12[LOCAL_OFFSET + 12
        + 9]) & 0xFF;

    l_12[0] = 12;
    memcpy(memory_12 + LOCAL_OFFSET + 12, input, 12);

    l_12[0] = 1;
    memory_12[LOCAL_OFFSET + 8] = input_brother;
```

```
    l_12[0] = 1;
    uint8_t return_brother = memory_12[LOCAL_OFFSET + 4];

    /* !trans12 */
    l_11[0] = 1;
    memory_11[LOCAL_OFFSET + 4] = return_brother;


    memory_11[LOCAL_OFFSET + 4] = (memory_11[LOCAL_OFFSET + 4] % 12) & 0
        xFF;
    memory_11[LOCAL_OFFSET + 8] = memory_11[LOCAL_OFFSET + 12 +
        memory_11[LOCAL_OFFSET + 4]];

    l_11[0] = 1;
    uint8_t ret_11 = l_11[2];

    /* trans 12 */

    memory_12[LOCAL_OFFSET + 8] = (memory_12[LOCAL_OFFSET + 8] % 12) & 0
        xFF;
    memory_12[LOCAL_OFFSET + 4] = memory_12[LOCAL_OFFSET + 12 +
        memory_12[LOCAL_OFFSET + 8]];

    l_12[0] = 1;
    uint8_t ret_12 = l_12[1];

    /* !trans 12 */

    return ret_11 ^ ret_12;
}
```