Solution du Challenge SSTIC 2015

Philippe Valembois

15 avril 2015

Résumé

Le challenge SSTIC consiste à trouver une adresse e-mail de la forme `xxx@challenge.sstic.org`. Cette année, le challenge commence par une image de carte microSD. Cela plongera les participants dans l'analyse d'une injection USB, puis vers une carte OpenArena, pour arriver ensuite sur une trace USB. La suite permettra d'étudier un programme JavaScript, une architecture exotique (le ST20) et enfin, de la stéganographie dans tous ses états.

Table des matières

1 Étude d'une carte microSD

Le challenge commence par l'étude d'une image de carte SD « insérée dans une clé USB étrange ». On commence par monter (en lecture seule) la carte SD avec la commande `mount -o loop,ro sdcard.img /mnt/`. L'image de la carte SD contient un fichier `inject.bin`.

La commande `file` nous indique un fichier inconnu :

```
$ file inject.bin
inject.bin: data
```

L'information étant maigre, il s'agit de trouver des informations supplémentaires éventuellement cachées dans l'image disque. Le logiciel TestDisk [1] est un candidat idéal pour cette tâche : TestDisk permet de récupérer des partitions perdues sur un disque ainsi que des fichiers supprimés récemment dans les systèmes de fichiers FAT, NTFS et ext2.

On exécute donc la commande `testdisk sdcard.img`.

Il s'agit d'une image sans table de partition ; dans le menu « Advanced », il faut sélectionner le mode « Undelete ». C'est gagné ! TestDisk trouve un fichier `build.sh` en plus du fichier `inject.bin`.

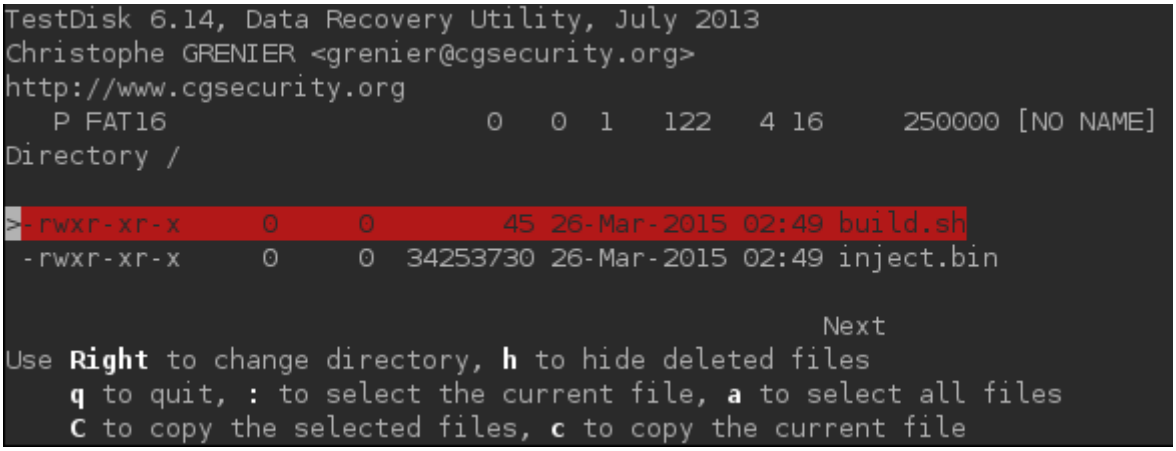

FIGURE 1 – TestDisk a retrouvé build.sh

Ce fichier contient la ligne :

```
java -jar encoder.jar -i /tmp/duckyscript.txt
```

> Il aurait été plus simple d'utiliser `strings sdcard.img` mais rien n'indiquait qu'un fichier contenant du texte serait caché.

Cette ligne de commande pointe, après une rapide recherche sur Internet, vers l'USB Rubber Ducky [2].

2 Extraction d'un script USB Rubber Ducky

L'USB Rubber Ducky se présente sous la forme d'une clé USB. Elle émule un clavier USB et injecte des commandes HID (qui informent que des touches ont été pressées) stockées sur une carte microSD. Cela corrobore donc l'indication initiale d'une clé USB étrange.

Le fichier de commandes de la Rubber Ducky est encodé par le programme Java `encode.jar`. Le résultat du programme est ensuite copié sur une carte microSD qui sera insérée dans la Rubber Ducky. Une fois la Rubber Ducky insérée dans un port USB, elle se présentera comme un clavier USB et enverra les séquences de touches précédemment enregistrées.

La phase d'encodage permet de limiter la complexité du micro-logiciel de la Rubber Ducky car il n'a pas à interpréter l'ensemble des constructions du langage de script.

Une lecture du code Java [3] et du micro-logiciel [4] présent dans la Rubber Ducky s'impose pour déterminer le format du fichier `inject.bin`.

1. http://www.cgsecurity.org/wiki/TestDisk
2. http://www.usbrubberducky.com/
3. https://github.com/hak5darren/USB-Rubber-Ducky/blob/master/Encoder/src/Encoder.java
4. https://github.com/hak5darren/USB-Rubber-Ducky/blob/master/Firmware/Source/Duck_HID/src/main.c

Chaque instruction du fichier est composée de deux octets.

— Si le premier octet est nul, le deuxième octet contient le délai à attendre avant de lire la prochaine instruction.

— Sinon, il s'agit d'un code de touche tel que documenté dans la HUT [5] concernant les claviers (Section 10). Le deuxième octet est alors un champ de bits contenant les modificateurs (Ctrl, Maj, OS, ...) tels que décrits dans la norme HID [6] (Section 8.3).

Pour interpréter le fichier `inject.bin`, le script `derubber.py` (annexe A), décodant les instructions Rubber Ducky, a été réalisé.

> Il existe un script `ducky-decode.pl` [7] qui fait exactement le même travail de manière exhaustive mais l'auteur admet avoir codé sans faire un état de l'art auparavant.

Le script permet d'obtenir la séquence de touches suivante :

```
Win+ r
cmd
powershell -enc ZgB1AG4AYwB0AGkAbwBuACAAdwByAGk...ApADsAfQA=
powershell -enc ZgB1AG4AYwB0AGkAbwBuACAAdwByAGk...ApADsAfQA=
...
```

Le Rubber Ducky une fois branché sur Windows, ouvrirait la boite de dialogue « Exécuter »(grâce à la combinaison de touche `Win+r`), lancerait une invite de commandes (`cmd`), et saisirait une série de commandes PowerShell. Chaque commande powershell est de la forme :

```
powershell -enc <Données encodées en Base64>
```

L'argument `-enc` de PowerShell permet d'exécuter le script suivant l'argument après décodage du Base64. Le script est de plus encodé en UTF-16.

Le *one-liner* Bash ci-dessous permet d'extraire les scripts PowerShell saisis dans l'invite de commande. Il consiste à isoler la partie encodée en Base64, la décoder et la convertir en UTF-8.

```
python derubber.py|
tail -n +3|
  (while read l; do
    echo "$l"|
      cut -b17-|
      base64 -d|
      iconv -f=utf16 -t=utf8;
    echo "";
  done
)> injection_scripts1.txt
```

Le résultat est un ensemble de scripts PowerShell identiques tels que mis en forme ci-après :

```
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('Ici un long blob Base64'));
}
else{
  write_file_bytes(
    [Convert]::FromBase64String('VAByAHkASABhAHIAZABlAHIA'));
}
```

5. HID Usage Table : http://www.usb.org/developers/hidpage/Hut1_12v2.pdf
6. Human Interface Device : http://www.usb.org/developers/hidpage/HID1_11.pdf
7. https://code.google.com/p/ducky-decode/source/browse/trunk/ducky-decode.pl

Ce script PowerShell vérifie dans un premier temps l'environnement d'exécution en concaténant la dernière composante du répertoire courant avec le nom de l'utilisateur. Si le résultat de la concaténation correspond à `challenge2015sstic`, un morceau de texte décodé depuis le blob Base64 est écrit dans le fichier `stage2.zip`. Sinon, le texte `TryHarder` sera écrit à la place.

Plutôt que d'exécuter le script, il est plus simple de couper astucieusement chaque ligne pour extraire les blobs Base64 et les décoder. Un one-liner shell convient parfaitement parfaitement :

```
cat injection_scripts.txt|
  (while read l; do
    echo "$l"
      |cut -b434-
      |rev
      |cut -b86-
      |rev
      |base64 -d;
   done)
> stage2.zip
```

On obtient alors le fichier `stage2.zip`. Ce dernier contient trois fichiers : `encrypted`, `memo.txt` et `sstic.pk3`.

```
$ unzip stage2.zip
Archive:  stage2.zip
 extracting: encrypted
  inflating: memo.txt
  inflating: sstic.pk3
```

Le fichier `memo.txt` contient des informations sur la résolution de cette étape :
— l'algorithme de chiffrement utilisé pour chiffrer `encrypted` : AES-OFB,
— le vecteur d'initialisation : `0x535354494332303135 2d537461676532`,
— le condensat SHA256 du fichier chiffré : `encrypted`,
— le condensat SHA256 du fichier déchiffré : `decrypted`,
La clé n'est pas donnée ; il est indiqué qu'elle est cachée dans le jeu préféré de l'auteur. Il s'agit alors d'analyser le fichier `sstic.pk3`.

3 Analyse d'une map OpenArena

La commande `file` indique que `sstic.pk3` est une archive ZIP. Une recherche sur Internet indique en sus que cette extension est associée aux cartes pour le jeu Quake III et ses dérivés.

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

La première étape consiste à décompresser fichier ZIP. L'archive extraite contient les éléments suisvants :

```
.  ..  AUTHORS  levelshots  maps  README  scripts  sound  textures
```

avec :
— `AUTHORS` : informations sur le créateur de l'archive ;
— `levelshots` : dossier contnant quelques captures d'écran pour le sélecteur de carte ;
— `maps` : les fichiers décrivant les cartes de l'archive ;
— `README` : un fichier à lire ;
— `scripts` : une description des modes de jeu des cartes ;
— `sound` : un dossier contenant les sons utilisés dans la carte ;
— `textures` : un dossier contenant les textures.
Le fichier `README` contient le texte suivant :

```
Copy the pk3 in your baseoa directory.
In the game, open the console and type \map sstic.
```

La première ligne indique qu'il faut copier le fichier pk3 dans un répertoire `baseoa`. Un recherche sur Internet indique que le répertoire `baseoa` correspond à une installation du jeu OpenArena.

Une petite exploration des différents répertoires montre que le répertoire `textures` contient des textures liées au challenge (108 pour être précis). Ces textures sont de deux types :

FIGURE 2 – Les deux types de textures du challenge

— des carrés contenant trois bouts de clés avec des couleurs orange, blanche et verte ainsi qu'un petit symbole (drapeau, goutte, maillon, ...) (80 éléments),
— des rectangles avec un carré de couleur orange, blanche ou verte et un symbole (24 éléments).

Pour le rectangle avec un carré orange et un symbole d'onde sonore, il est aisé de comprendre qu'il faut prendre la partie de clé de couleur orange sur une texture avec le même symbole. Deux problèmes se posent alors : l'ordre dans lequel enchainer les fragments de clé et la multiplicité des possibilités. En effet, les symboles sont répétés dans plusieurs tuiles.

La résolution du challenge passera donc par l'installation de OpenArena [8]. S'ensuivra une exploration de la carte dans le jeu qui permettra de remarquer certaines textures repérées précedemment ainsi que différentes actions possibles pour accéder à des parties cachées.

Un éditeur de cartes Quake III sera d'une aide précieuse pour se repérer dans le jeu. Le logiciel GtkRadiant [9] fournit ce type de fonctionnalités et permettra d'étudier la carte du challenge dans tous ses détails.

GtkRadiant ne permet pas charger la carte directement car celle-ci a été compilée. Il faut d'abord passer par une étape de décompilation.

```
$ q3map2 -game oa -convert -format map maps/sstic.bsp
2.5.17
threads: 4
Q3Map         - v1.0r (c) 1999 Id Software Inc.
Q3Map (ydnar) - v2.5.17
GtkRadiant    - v1.6.4 Dec 13 2013 17:18:47
We're still here
VFS Init: /.q3a/baseq3/
VFS Init: /opt//baseq3/

entering scripts/shaderlist.txt
...
Loading /stage2/pk3/maps/sstic.bsp
--- Convert BSP to MAP ---
writing /stage2/pk3/maps/sstic_converted.map
...
WARNING: Couldn't find image for shader textures/dga/691105128
WARNING: Couldn't find image for shader textures/dga/747908186
WARNING: Couldn't find image for shader textures/dga/1579532252
...
        1 seconds elapsed
```

Le fichier `sstic_converted.map` est créé et il est ouvrable dans GtkRadiant. Après une prise en main du logiciel, il est possible découvrir une salle cachée. Autour de cette salle cachée, il est possible de voir des lignes noires sur le plan. Il s'agit des tuiles en surplus qui sont placées dans un endroit inaccessible de la carte. Cela évite de filtrer les tuiles invalides en étudiant la liste des textures référencées par la carte.

8. http://www.openarena.ws/
9. http://icculus.org/gtkradiant/

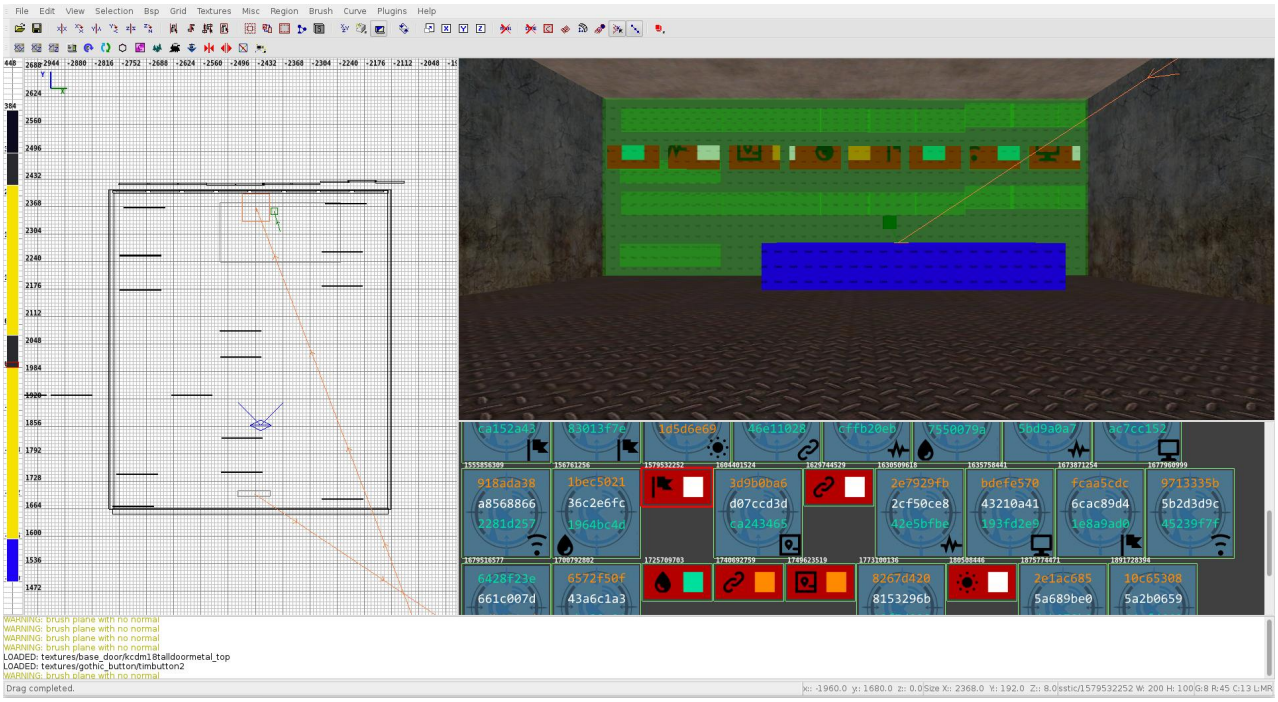

FIGURE 3 – La salle cachée telle que vue dans GtkRadiant

Une fois ces informations collectées, il reste à se promener dans la carte pour trouver les textures réellement utilisées et les associations. Pour accéder facilement à la salle cachée, il est possible d'activer un mode de triche dans OpenArena. Il suffit d'ouvrir la console du jeu et de saisir :

```
\devmap sstic
\noclip
```

Le joueur se retrouve à flotter dans l'environnement et peut passer à travers les murs. Il est alors très simple de récupérer l'ensemble des informations.

FIGURE 4 – Les différentes informations nécessaires au déchiffrement

L'ensemble des images permet de reconstituer la clé de déchiffrement. En effet, pour chaque paire symbole/couleur de l'image du bas, il faut choisir la tuile associée au symbole et le morceau de clé correspondant à la couleur.

— Drapeau vert : `9e2f31f7`;
— Onde blanche : `8153296b`;
— Checkpoint orange : `3d9b0ba6`;
— Goutte blanc : `7695dc7c`;
— Drapeau orange : `b0daf152`;
— Chaine vert : `b54cdc34`;
— Wifi vert : `ffe0d355`;
— Ecran blanc : `26609fac`;

La clé de chiffrement du fichier `encrypted` est donc :

```
9e2f31f78153296b3d9b0ba67695dc7cb0daf152b54cdc34ffe0d35526609fac
```

Il suffit ensuite de lancer la commande :

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

Une vérification du condensat avec la commande `sha256sum` aboutit à un échec : le résultat est différent de celui renseigné dans `memo.txt`. Pourtant, le fichier semble valide car `file` reconnait un fichier ZIP.

```
$ sha256sum decrypted
f9ca4432afe87cbb1fca914e35ce69708c6bfa360b82bff21503b6723d1cfbf0  decrypted
$ file decrypted
decrypted: Zip archive data, at least v1.0 to extract
```

En fait, le fichier `decrypted` est terminé par un bourrage PKCS#7 qui n'est pas supprimé par OpenSSL. Le mode de chiffrement OFB transformant AES en chiffrement par flot rendait inutile l'utilisation d'un bourrage. Après suppression du padding à l'aide d'un éditeur hexadécimal, le condensat est vérifié.

```
$ sha256sum decrypted
845f8b000f70597cf55720350454f6f3af3420d8d038bb14ce74d6f4ac5b9187  decrypted
```

Il faut ensuite décompresser ce fichier `decrypted`. Il contient trois fichiers `encrypted`, `memo.txt` et `paint.cap`.

```
$ unzip decrypted
Archive: decrypted
 extracting: encrypted
  inflating: memo.txt
  inflating: paint.cap
```

Le fichier `memo.txt` contient des informations sur la résolution de cette étape :
— l'algorithme de chiffrement utilisé pour chiffrer `encrypted` : Serpent-1-CBC-With-CTS,
— le vecteur d'initialisation : `0x53535449433230313 52d537461676533`,
— le condensat SHA256 du fichier chiffré : `encrypted`,
— le condensat SHA256 du fichier déchiffré : `decrypted`,

Le fichier indique aussi que la clé a été stockée avec Paint. Le prochain fichier à être étudier est donc `paint.cap`.

4 Récupération d'une clé dessinée

4.1 Etude de la capture

La commande `file` exécutée sur le fichier `paint.cap` indique qu'il s'agit d'une capture obtenue grâce à `tcpdump` et qu'elle contient des trames USB.

```
$ file paint.cap
paint.cap: tcpdump capture file (little-endian) - version 2.4 (Memory-mapped
↪ Linux USB, capture length 262144)
```

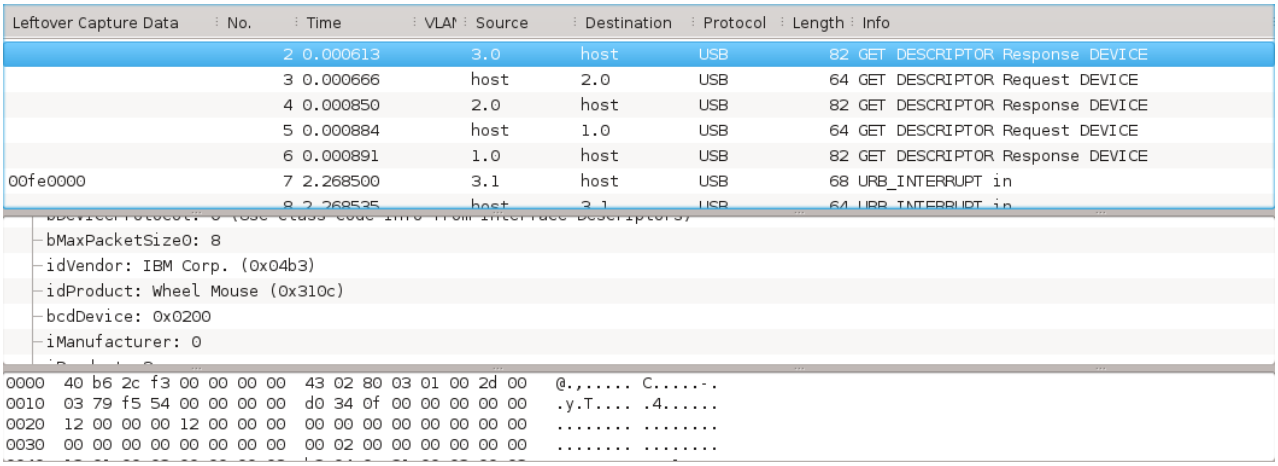

FIGURE 5 – Le fichier `paint.cap` affiché dans Wireshark

Wireshark permet de décoder ce fichier et d'afficher des détails sur le protocole USB.

Le paquet `GET DESCRIPTOR Response DEVICE` montre qu'il s'agit de la capture des paquets envoyés par une souris USB. Les paquets suivants sont des requêtes-réponses de type `INTERRUPT`. Ces divers éléments laissent supposer qu'il s'agit d'un HID[10] *report* tel que décrit dans la spécification à la section 8.

Le fichier `memo.txt` mentionnant le logiciel Paint et cette capture de données provenenant d'une souris, il est probable que la clé ait été dessinée dans Paint à l'aide de cette souris. En retrouvant les mouvements effectués, il sera possible de retrouver le dessin effectué et de fait, la clé.

La spécification décrit le format générique du *report*. En effet, chaque périphérique est libre de personnaliser ses messages. Lors du branchement d'un périphérique HID, l'hôte USB va interroger le périphérique avec un message de type `GET DESCRIPTOR Request` qui a son champ `Request Type` placé à `0x22` (Section 7.1 de la spécification). Le périphérique va alors répondre avec une structure décrivant les messages qu'il enverra dans ses *reports* (Section 5.2). Cet échange est cependant manquant dans la capture. La spécification HID fournit des exemples de `report descriptors` (Sections 8.5, B.2 et E.10) qui décrivent des formats de message possibles. En analysant les différents paquets, il est possible d'arriver au format de message suivant.

0	1	2	3	octets
Boutons	X	Y	Z(?)	

FIGURE 6 – Le format des *reports* HID utilisés par des souris

Les champs X et Y décrivent la position de la souris en coordonnées relatives par rapport au dernier message. Le champ « Boutons »est composé de trois bits – les moins significatifs – qui décrivent l'état des trois boutons de la souris (1 pour enfoncé, 0 pour relaché). Le champ Z étant toujours à 0, il s'agit là d'une supposition. Il ne sera de toute manière pas utilisé.

4.2 Extraction et traitement des données

L'extraction des données se fait avec Wireshark. Le champ `Leftover Capture Data` est ajouté aux colonnes puis la commande « Export Packets Dissections as CSV »est utilisée. Un éditeur de texte est ensuite utilisé pour ne garder que le champ `Leftover Capture Data`.

Une fois cette opération menée à bien, il faut reproduire les mouvements de souris qui ont été effectués. Ces mouvements sont codés à l'aide de coordonnées X et Y relatives. Le script Python `draw_mouse.py` (annexe B) va permettre de reconstruire l'image.

Le script crée une image de 4096x4096 pixels et, en partant du centre, va dessiner un point pour chaque message. Le point sera dessiné en rouge si un bouton de la souris était enfoncé, en noir sinon (Figure 7).

On peut donc lire la clé de chiffrement : « The quick brown fox jumps over the lobster dog »qui sera hashée avec l'algorithme `Blake-256`.

10. Human Interface Device : http://www.usb.org/developers/hidpage/HID1_11.pdf

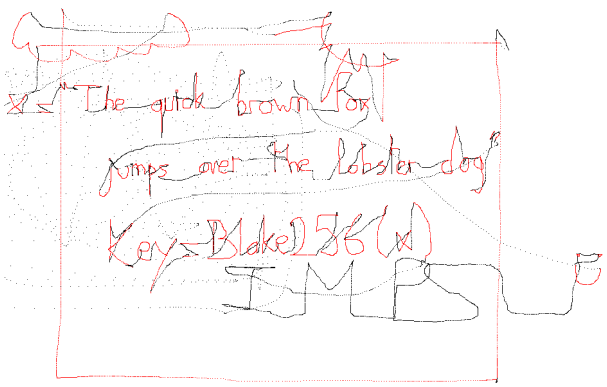

FIGURE 7 – L'image résultante

4.3 Déchiffrement du fichier `encrypted`

L'algorithme `Blake-256` n'est pas présent dans OpenSSL. C'est un candidat malheureux à la compétition SHA-3 organisée par le NIST. Il faut donc récupérer l'implémentation de référence [11].

```
$ echo -n "The quick brown fox jumps over the lobster dog" | ./blake256
↪  /dev/stdin
66c1ba5e8ca29a8ab6c105a9be9e75fe0ba07997a839ffeae9700b00b7269c8d /dev/stdin
```

On obtient alors la clé de déchiffrement. Il reste cependant une difficulté supplémentaire : retrouver l'algorithme de chiffrement utilisé et son mode de fonctionnement. Une recherche sur Internet permet de trouver la bibliothèque `Crypto++` qui possède une implémentation de `Serpent-1` et du mode `CBC-With-CTS`.

Le programme effectuant un déchiffrement Serpent-1-CBC-With-CTS (annexe C) est lancé. Le résultat est concluant car le condensat du fichier `decrypted` correspond au condensat renseigné dans le fichier `memo.txt`.

```
$ sha256sum decrypted
7beabe40888fbbf3f8ff8f4ee826bb371c596dd0cebe0796d2dae9f9868dd2d2  decrypted
```

Il s'agit d'un fichier ZIP à décompresser qui contient un unique fichier `stage4.html`.

```
$ unzip decrypted
Archive:  decrypted
  inflating: stage4.html
```

Le fichier `stage4.html` comporte un gros bloc de code JavaScript obscurci. Il va donc falloir l'étudier de plus près.

5 Éclaircissement d'un code JavaScript

Le fichier `stage4.html` se présente sous la forme suivante :

```
<html>
<head>
<style>
        * { font-family: Lucida Grande,Lucida Sans Unicode,Lucida Sans,Geneva,Verdana,sans-serif;
        ↪  text-align:center; }
        #status { font-size: 16px; margin: 20px; }
        #status a { color: green; }
        #status b { color: red; }
</style>
</head>
<body>
        <script>
          var data = "2b1f25cf8db5d243f59b065da6b56753b72...";
          var hash = "08c3be636f7dffd91971f65be4cec3c6d162cb1c";
          $=~[];$={___:++$,$$$$:(![]+"")[$],__$:++$,$_$_:(![]+"")[$],_$_:++$,$_$$...;
        </script>
```

11. https://131002.net/blake/blake_c.tar.gz

```
</body>
</html>
```

Le code JavaScript est composé de trois parties : la définition de `data`, celle de `hash` et enfin, le code obscurci. Une fois passé dans un embellisseur de code JavaScript [12], le code est à peine plus compréhensible.

```
$ = ~[];
$ = {
    ___: ++$,
    $$$$: (![] + "")[$],
    __$: ++$,
    $_$_: (![] + "")[$],
    _$_: ++$,
    $_$$: ({} + "")[$],
    $$_$: ($[$] + "")[$],
    _$$: ++$,
    $$$_: (!"" + "")[$],
    $__: ++$,
    $_$: ++$,
    $$__: ({} + "")[$],
    $$_: ++$,
    $$$: ++$,
    $___: ++$,
    $__$: ++$
};
$.$_ = ($.$_ = $ + "")[$.$_$]...;
$.$$ = $.$ + (!"" + "")[$._$$] + $.__ + $._ + $.$ + $.$$;
$.$ = ($.___)[$.$_][$.$_];
$.$($.$($.$$ + "\"" ... "\\" + $.__$ + $.__$ + "\"")())();
```

Il faut passer par le dévermineur JavaScript d'un navigateur (Chromium, par exemple) pour mieux comprendre l'enchainement du script. La ligne la plus intéressante est la dernière. En effet, les lignes précédentes ne servent qu'a préparer l'appel final en construisant `$.$` et les autres variables. `$.$` est l'équivalent de la fonction `eval`. Il est construit de la manière suivante :

```
$.$ = 0["constructor"]["constructor"]
```

Cette construction renvoie le constructeur de `Function` qui prend en argument le code source de la fonction à instancier. Pour aller plus loin, il est nécessaire de modifier le fonctionnement du script. Pour cela, entre l'affectation de `$.$` et ses appels, il faut redéfinir, grâce à la console du navigateur, `$.$`.

```
toto = $.$; $.$ = function() { debugger; return toto.apply(this, arguments); };
```

Cette redéfinition déclenchera un point d'arrêt lors de l'exécution de la fonction. Il devient alors possible d'inspecter les arguments de l'appel.

```
return"__=docu\155..."
```

Dans le code ci-dessus, il était possible de voir que `$.$` était appelé deux fois. Le premier appel ne donne pas un code très intéressant ni très propre. Le deuxième appel est déjà beaucoup plus intéressant. Le voici une fois embelli.

```
__ = document;
...
$$_$$$ = 'navigator';
...
$_$$$$ = 'Agent';
...
__$ = $$.crypto.subtle;
__$_ = 'decrypt';
___$ = 'status';
$____ = $$$$_$ + 'Key';
...
___$__ = {
    name: 'SHA-1'
```

12. http://jsbeautifier.org/

```
};
...
__$_____ = 'AES-CBC';
___$____ = 'iv';
...

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

Le code étant encore trop obscur, l'utilisation du dévermineur JavaScript d'un navigateur est indispensable. Pour cela, le code nouvellement obtenu est mis en lieu et place de l'ancien code obscurci dans le fichier HTML.

Le code est en fait simple, il fait appel à l'API cryptographique du navigateur pour déchiffrer la variable data à l'aide de l'algorithme de chiffrement AES en mode CBC et vérifier le condensat à l'aide de l'algorithme SHA-1.

La clé et le vecteur d'initialisation sont extraits du champ `User-Agent` du navigateur exécutant le JavaScript. Les 16 premiers octets après la première parenthèse ouvrante servent de vecteur d'initialisation et les 16 derniers octets avant la première parenthèse fermante servent de clé.

Il faut donc extraire le contenu de la variable `data` dans un fichier `encrypted` et le transformer en octets binaires à l'aide de la commande `xxd -r -p`. Ensuite, un script Python permettra d'effectuer la même tâche de déchiffrement que le code JavaScript. Il enlèvera le bourrage et il vérifiera le condensat pour déterminer si la clé et le vecteur d'initialisation sont valides.

Il faut ensuite nourrir ce script à l'aide d'une liste de champs `User-Agent` exhaustive. La liste `http://www.ua-tracker.com/user_agents.txt` contient plus de 83000 entrées.

Le script est lancé et obtient une solution. Le User-Agent `Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:35.0) Gecko/20100101 Firefox/35.0` permet de déchiffrer le fichier.

Un fichier `decrypted` est obtenu. C'est un fichier ZIP contenant deux fichiers : `schematic.pdf` et `input.bin`.

```
$ unzip decrypted
Archive: decrypted
  inflating: input.bin
  inflating: schematic.pdf
```

Le fichier `input.bin` contient de la donnée.

```
$ file input.bin
input.bin: data
```

Le fichier `schematic.pdf` contient :
— un schéma de ce qui semble être un ensemble d'automates,
— le hash d'un fichier `encrypted` non présent,
— le hash d'un fichier `decrypted`,
— un jeu de test pour un algorithme de chiffrement.

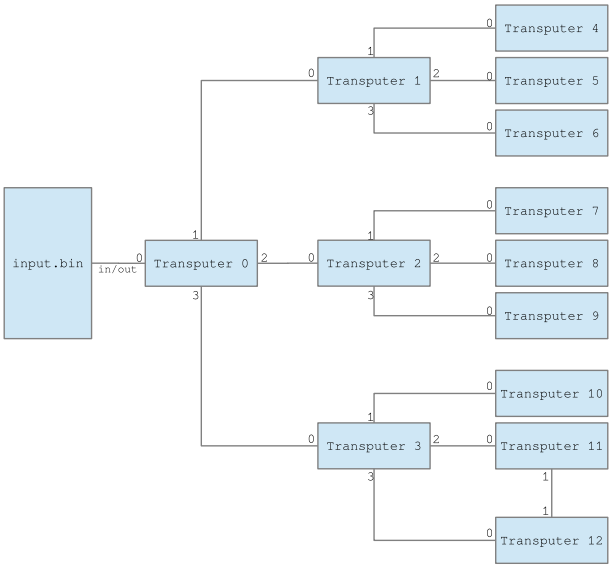

FIGURE 8 – Schéma inclus dans `schematic.pdf`

Le jeu de test emploie la phrase « I love ST20 architecture »comme texte clair. Cela semble donc indiquer que le fichier `input.bin` contient du code machine pour une architecture ST20. De plus, la mention de "transputer" dans le schéma corrobore cette supposition. La suite du challenge consistera donc à analyser ce fichier.

6 Ingénierie à reculons d'une architecture multi-cœurs ST20

6.1 Etude de l'architecture ST20

L'architecture ST20 est inspirée d'une architecture ancienne (datant des années 80) développée à l'origine par la société Inmos et rachetée ensuite par STMicroelectronics. Sa spécificité était de posséder plusieurs cœurs reliés entre eux par un bus série ; c'est ce que le schéma ci-dessus présente.

Le jeu d'instruction est disponible sur entre autres sur un site dédié aux architectures à base de transputeur [13].

6.2 Rétro-ingénierie de `input.bin`

6.2.1 Format du binaire

Le fichier `input.bin` est composé de plusieurs parties :
— le code des transputeurs ;
— l'emplacement pour la clé de déchiffrement ;
— le binaire à déchiffrer.

L'ensemble du fichier est envoyé par un élément extérieur au microprocesseur vers le transputeur 0 qui se chargera de traiter les données.

La partie à déchiffrer est isolée pour permettre une rétro-ingénierie plus aisée. Pour cela, le script Python ci-dessous suppose que le contenu est placé à la fin du fichier et itère depuis le début en utilisant le condensat fourni comme condition d'arrêt.

13. http://www.transputer.net/iset/pdf/st20core.pdf

```
import hashlib
import sys

data = file("input.bin", "rb").read()

for i in range(len(data)):
        hasher = hashlib.sha256()
        hasher.update(data[i:])
        dig = hasher.hexdigest()
        if dig == "a5790b4427bc13e4f4e9f524c684809ce96cd2f724e29d94dc999ec25e166a81":
                print i
                file("encrypted", "wb").write(data[i:])
                sys.exit(0)
```

FIGURE 9 – Programme extrayant les données chiffrées depuis le binaire

6.2.2 Etude des parties de chargement de code

Les transputeurs sont chargés séquentiellement. Le transputeur 0 est le premier à être chargé. Une documentation sur les transputeurs explique que le chargement du code se fait sur le premier lien série du micro-controleur [14]. Le démarreur lit un premier octet indiquant la taille du programme à charger et lit ensuite les octets du code.

Une fois le code du transputeur 0 chargé, celui-ci est responsable de distribuer le code aux autres transputeurs.

Le code lit un bloc de 12 octets contenant la taille du code à charger ainsi que l'adresse du canal où envoyer les données.

0	4	8	octets
Taille du code	Adresse du canal	Inutilisé	

FIGURE 10 – Le format de l'en-tête avant le code des autres transputeurs

Si le champ taille vaut 0, le chargement du code est terminé et le transputeur 0 commence le déchiffrement.

Les transputeurs 1, 2 et 3 ont le même mécanisme pour charger le code des transputeurs 4 à 12.

14. http://www.transputer.net/tn/34/tn34.html

```
ROM:0000 F8                         db 0F8h                 ; Le code à copier fait 0xF8 bytes
ROM:0001
ROM:0001             transputer0:
ROM:0001 64 B4                      ajw     -76
ROM:0003 40                         ldc     0
ROM:0004 D1                         stl     1
ROM:0005 40                         ldc     0
ROM:0006 D3                         stl     3
ROM:0007 24 F2                      mint
ROM:0009 24 20 50                   ldnlp   400h
ROM:000C 23 FC                      gajw
ROM:000E 64 B4                      ajw     -76
ROM:0010 2C 49                      ldc     0C9h ; '+'
ROM:0012 21 FB                      ldpi                    ; C9+14 = DD
ROM:0014 24 F2                      mint
ROM:0016 48                         ldc     8
ROM:0017 FB                         out                     ; ecrit Boot OK vers le canal
 ↪  0x80000000
ROM:0018
ROM:0018             loc_18:                                ; CODE XREF: transputer0+35
ROM:0018 24 19                      ldlp    49h ; 'I'
ROM:001A 24 F2                      mint
ROM:001C 54                         ldnlp   4
ROM:001D 4C                         ldc     0Ch
ROM:001E F7                         in                      ; lit 12 octets depuis le canal
 ↪  0x8000010 vers local.49
ROM:001F
ROM:001F             loc_1F:
ROM:001F 24 79                      ldl     49h ; 'I'
ROM:0021 21 A5                      cj      loc_38          ; saute si le premier mot lu est 0
ROM:0023 2C 4D                      ldc     0CDh ; '-'
ROM:0025 21 FB                      ldpi                    ; CD+27 = F4
ROM:0027 24 F2                      mint
ROM:0029 54                         ldnlp   4               ; 0x80000010
ROM:002A 24 79                      ldl     49h ; 'I'
ROM:002C F7                         in                      ; lit [local.49] octets depuis le
 ↪  canal 0x8000010 vers 0xF4
ROM:002D 2C 43                      ldc     0C3h ; '+'
ROM:002F 21 FB                      ldpi                    ; C3+31 = F4
ROM:0031 24 7A                      ldl     4Ah ; 'J'
ROM:0033 24 79                      ldl     49h ; 'I'
ROM:0035 FB                         out                     ; Envoie [local.49] octets vers le
 ↪  canal [local.4A] depuis 0xF4
ROM:0036 61 00                      j       loc_18
ROM:0038             ; ---------------------------------------------------------------------------
ROM:0038
ROM:0038             loc_38:                                ; CODE XREF: transputer0+20
ROM:0038 24 19                      ldlp    49h ; 'I'
ROM:003A 24 F2                      mint
ROM:003C 51                         ldnlp   1               ; 0x80000004
ROM:003D 4C                         ldc     0Ch
ROM:003E FB                         out                     ; Envoie 12 octets vers le canal
 ↪  0x80000004 depuis local.49
ROM:003F 24 19                      ldlp    49h ; 'I'
ROM:0041 24 F2                      mint
ROM:0043 52                         ldnlp   2
ROM:0044 4C                         ldc     0Ch
ROM:0045 FB                         out                     ; Envoie 12 octets vers le canal
 ↪  0x80000008 depuis local.49
ROM:0046 24 19                      ldlp    49h ; 'I'
ROM:0048 24 F2                      mint
ROM:004A 53                         ldnlp   3
ROM:004B 4C                         ldc     0Ch
ROM:004C FB                         out                     ; Envoie 12 octets vers le canal
 ↪  0x8000000C depuis local.49
ROM:004D 29 44                      ldc     94h ; 'ö'
ROM:004F 21 FB                      ldpi                    ; 94+51 = E5
ROM:0051 24 F2                      mint
ROM:0053 48                         ldc     8
ROM:0054 FB                         out                     ; Envoie Code OK vers le canal
 ↪  0x80000000
```

FIGURE 11 – Code de chargement du transputeur 0

Le chargement de code des transputeur 4 à 12 se fait en deux étapes. Une première étape lance un chargeur de code plus évolué que le chargeur de base. En effet, ce dernier est capable de démarrer le deuxième étage de code à partir d'une adresse arbitraire. Le format de l'en-tête est donc revu :

0	4	8	octets
Taille du code	Inutilisé	Adresse de démarrage	

FIGURE 12 – Le format de l'en-tête avant le code final des transputeurs 4 à 12

```
ROM:0270 31 00 00 00                dd 31h                  ; Taille pour le transputer 0
ROM:0274 04 00 00 80                dd 80000004h            ; Canal 0x80000004
ROM:0278 00 00 00 00                dd 0
ROM:027C 25 00 00 00                dd 25h                  ; Taille pour le transputer 1
ROM:0280 04 00 00 80                dd 80000004h            ; Canal 0x80000004
ROM:0284 00 00 00 00                dd 0
ROM:0288 24                         db 24h                  ; Taille pour le démarreur du
 ↪  transputer 4
ROM:0289
ROM:0289            ; ============== S U B R O U T I N E =======================================
ROM:0289
ROM:0289
ROM:0289            transputer41:
ROM:0289 60 BD                      ajw     -3
ROM:028B 24 F2                      mint
ROM:028D 24 20 50                   ldnlp   400h
ROM:0290 23 FC                      gajw
ROM:0292 60 BD                      ajw     -3
ROM:0294 10                         ldlp    0
ROM:0295 24 F2                      mint
ROM:0297 54                         ldnlp   4
ROM:0298 4C                         ldc     0Ch
ROM:0299 F7                         in                      ; Lit 12 octets depuis le canal
 ↪  0x80000004 vers local.0
ROM:029A 4B                         ldc     0Bh
ROM:029B 21 FB                      ldpi                    ; 29D + 0B = 2A8
ROM:029D 24 F2                      mint
ROM:029F 54                         ldnlp   4
ROM:02A0 70                         ldl     0
ROM:02A1 F7                         in                      ; Lit [local.0] octets depuis le canal
 ↪  0x80000004 vers 0x2A8 (juste a la fin du code actuel)
ROM:02A2 43                         ldc     3
ROM:02A3 21 FB                      ldpi                    ; 2A5+3 = 2A8
ROM:02A5 72                         ldl     2
ROM:02A6 F2                         bsub                    ; 0x2A8 + [local.2]
ROM:02A7 F6                         gcall                   ; On appelle le code droppé
ROM:02A8 00                         j       loc_2A9
ROM:02A9
ROM:02A9            loc_2A9:
ROM:02A9 B3                         ajw     3
ROM:02AA 22 F0                      ret
ROM:02AA            ; End of function transputer41
```

FIGURE 13 – Code de chargement du transputeur 4

Le code du transputeur 4 consiste à lire 12 octets de clé depuis son canal 0. Il ajoute ensuite chaque octet à une variable interne modulo 256 et il sort un octet vers le transputeur 1.

```
ROM:0495 5C 00 00 00                dd 5Ch                  ; Taille pour le transputer 0
ROM:0499 04 00 00 80                dd 80000004h            ; Canal 0x80000004 pour le transputer 0
ROM:049D 0C 00 00 00                dd 0Ch                  ; Adresse de démarrage (inutilisé)
ROM:04A1 50 00 00 00                dd 50h                  ; Taille pour le transputeur 1
ROM:04A5 04 00 00 80                dd 80000004h            ; Canal 0x80000004 pour le transputer 1
ROM:04A9 0C 00 00 00                dd 0Ch                  ; Adresse de démarrage (inutilisé)
ROM:04AD 44 00 00 00                dd 44h                  ; Taille pour le transputeur 4
ROM:04B1 00 00 00 00                dd 0                    ; Canal inutilisé
ROM:04B5 0C 00 00 00                dd 0Ch                  ; Adresse de démarrage
ROM:04B9
ROM:04B9            ; ============== S U B R O U T I N E =======================================
ROM:04B9
ROM:04B9            ; buffer
ROM:04B9
ROM:04B9            read_chan2i:                            ; CODE XREF: transputer4+F^^Yp
ROM:04B9 73                         ldl     3
ROM:04BA 72                         ldl     2               ; channel
ROM:04BB 74                         ldl     4               ; size
ROM:04BC F7                         in
ROM:04BD 22 F0                      ret
ROM:04BD            ; End of function read_chan2i
ROM:04BD
ROM:04BF
ROM:04BF            ; ============== S U B R O U T I N E =======================================
ROM:04BF
ROM:04BF
ROM:04BF            write_chan2i:                           ; CODE XREF: transputer4+30^^Yp
ROM:04BF 73                         ldl     3
ROM:04C0 72                         ldl     2               ; channel
ROM:04C1 74                         ldl     4               ; size
ROM:04C2 FB                         out
ROM:04C3 22 F0                      ret
ROM:04C3            ; End of function write_chan2i
ROM:04C3
ROM:04C5            ; ============== S U B R O U T I N E =======================================
ROM:04C5
ROM:04C5            transputer4:
ROM:04C5 60 BB                      ajw     -5
ROM:04C7 40                         ldc     0
ROM:04C8 D1                         stl     1               ; local.1 = 0
ROM:04C9 40                         ldc     0
ROM:04CA 11                         ldlp    1
ROM:04CB 23 FB                      sb                      ; local.1 = 0
ROM:04CD            loc_4CD:                                ; CODE XREF: transputer4+32^^Yj
ROM:04CD 4C                         ldc     0Ch
ROM:04CE D0                         stl     0               ; local.0 = 12
ROM:04CF 12                         ldlp    2
ROM:04D0 24 F2                      mint
ROM:04D2 54                         ldnlp   4
ROM:04D3 76                         ldl     6
ROM:04D4 61 93                      call    read_chan2i     ; Lit 12 octets from 0x80000010 in local.2
ROM:04D6 40                         ldc     0
ROM:04D7 D0                         stl     0               ; local.0 = 0
ROM:04D8
ROM:04D8            loc_4D8:                                ; CODE XREF: transputer4+28^^Yj
ROM:04D8 70                         ldl     0
ROM:04D9 12                         ldlp    2
ROM:04DA F2                         bsub
ROM:04DB F1                         lb                      ; load local2[local0]
ROM:04DC 11                         ldlp    1
ROM:04DD F1                         lb
ROM:04DE F2                         bsub                    ; local2[local0] + local1
ROM:04DF 2F 4F                      ldc     0FFh
ROM:04E1 24 F6                      and                     ; (local2[local0] + local1) & 0xFF
ROM:04E3 11                         ldlp    1
ROM:04E4 23 FB                      sb                      ; local1 = (local2[local0] + local1) & 0xff
ROM:04E6 70                         ldl     0
ROM:04E7 81                         adc     1
ROM:04E8 D0                         stl     0               ; [local0]++
ROM:04E9 4C                         ldc     0Ch
ROM:04EA 70                         ldl     0
ROM:04EB F9                         gt
ROM:04EC A2                         cj      loc_4EF
ROM:04ED 61 09                      j       loc_4D8
ROM:04EF            ; ---------------------------------------------------------------------------
ROM:04EF            loc_4EF:                                ; CODE XREF: transputer4+27^^Xj
ROM:04EF 41                         ldc     1
ROM:04F0 D0                         stl     0
ROM:04F1 11                         ldlp    1
ROM:04F2 24 F2                      mint
ROM:04F4 76                         ldl     6
ROM:04F5 63 98                      call    write_chan2i    ; Envoie un octet à 0x80000000 depuis local.1
ROM:04F7 20 62 03                   j       loc_4CD
ROM:04F7            ; End of function transputer4
```

FIGURE 14 – Code du transputeur 4

```
def transputer4(inp):
        for i in range(12):
                transputer4.total = (transputer4.total + inp[i]) & 0xff
        return transputer4.total
transputer4.total = 0
```

FIGURE 15 – Code du transputeur 4 en Python

Les transputeurs 1, 2 et 3 servent de passe-plat et répartissent les 12 octets de clé vers chaque transputeur fils. Il fusionnent ensuite les résultats à l'aide de l'opérateur OU EXCLUSIF.

```
ROM:012C 11                         ldlp    1
ROM:012D 24 F2                      mint
ROM:012F 54                         ldnlp   4
ROM:0130 4C                         ldc     0Ch
ROM:0131 F7                         in                      ; Lit 12 octets depuis le canal
 ↪  0x80000010 vers local.1
ROM:0132 11                         ldlp    1
ROM:0133 24 F2                      mint
ROM:0135 51                         ldnlp   1
ROM:0136 4C                         ldc     0Ch
ROM:0137 FB                         out                     ; Ecrit 12 octets vers le canal
 ↪  0x80000004 depuis local.1
ROM:0138 11                         ldlp    1
ROM:0139 24 F2                      mint
ROM:013B 52                         ldnlp   2
ROM:013C 4C                         ldc     0Ch
ROM:013D FB                         out                     ; Ecrit 12 octets vers le canal
 ↪  0x80000008 depuis local.1
ROM:013E 11                         ldlp    1
ROM:013F 24 F2                      mint
ROM:0141 53                         ldnlp   3
ROM:0142 4C                         ldc     0Ch
ROM:0143 FB                         out                     ; Ecrit 12 octets vers le canal
 ↪  0x8000000C depuis local.1
ROM:0144 10                         ldlp    0
ROM:0145 81                         adc     1
ROM:0146 24 F2                      mint
ROM:0148 55                         ldnlp   5
ROM:0149 41                         ldc     1
ROM:014A F7                         in                      ; Lit un octet depuis le canal
 ↪  0x80000014 vers [local.0+1]
ROM:014B 10                         ldlp    0
ROM:014C 82                         adc     2
ROM:014D 24 F2                      mint
ROM:014F 56                         ldnlp   6
ROM:0150 41                         ldc     1
ROM:0151 F7                         in                      ; Lit un octet depuis le canal
 ↪  0x80000018 vers [local.0+2]
ROM:0152 10                         ldlp    0
ROM:0153 83                         adc     3
ROM:0154 24 F2                      mint
ROM:0156 57                         ldnlp   7
ROM:0157 41                         ldc     1
ROM:0158 F7                         in                      ; Lit un octet depuis le canal
 ↪  0x8000001C vers [local.0+2]
ROM:0159 10                         ldlp    0
ROM:015A 81                         adc     1
ROM:015B F1                         lb
ROM:015C 10                         ldlp    0
ROM:015D 82                         adc     2
ROM:015E F1                         lb
ROM:015F 23 F3                      xor                     ; transputer4 xor transputer5
ROM:0161 10                         ldlp    0
ROM:0162 83                         adc     3
ROM:0163 F1                         lb
ROM:0164 23 F3                      xor                     ; transputer4 xor transputer5 xor
 ↪  transputer6
ROM:0166 25 FA                      dup
ROM:0168 10                         ldlp    0
ROM:0169 23 FB                      sb                      ; [local.0+0] = [local.0+1] ^
 ↪  [local.0+2] ^ [local.0+3]
ROM:016B 10                         ldlp    0
ROM:016C 24 F2                      mint
ROM:016E 41                         ldc     1
ROM:016F FB                         out                     ; Ecrit un octet vers le canal
 ↪  0x80000000 depuis local.0
ROM:0170 64 0A                      j       loc_12C
```

FIGURE 16 – Code du transputeur 1, partie déchiffrement

Le transputeur 0 est responsable du déchiffrement en utilisant le flux chiffrant fourni par les autres transputeurs.

Son implémentation en Python pourrait s'écire comme ci-dessous :

```
def transputer0(key, data):
        key = list(key)
        i = 0
        result = []
        for c in data:
                tmp = ((key[i] * 2 + i) & 0xff) ^ ord(c)
                result.append(chr(tmp))

                msk = transputer1(key) ^ transputer2(key) ^ transputer3(key)
                key[i] = msk

                i = i + 1
                if i >= 12:
                        i = 0
        return "".join(result)
```

L'ensemble de l'implémentation est disponible en annexe D.

6.3 Cryptanalyse et cassage du chiffrement

Une analyse du transputeur 0, permet de trouver une faiblesse dans son fonctionnement. En effet, les octets de la clé initiale sont directement utilisés pour chiffrer les 12 premiers octets de texte clair.

```
  plain = ((key[i] * 2 + i) & 0xff) ^ ciphered
```

On voit ici qu'il est possible de renverser la formule pour trouver `key`. Il manquera par contre le bit de poids fort de chaque octet de clé car il a été perdu lors de la multiplication par 2.

```
        b = ord(clear[i]) ^ ord(cipher[i])
        if b < i:
                b = b + 0x100
        b = b - i
        b = b / 2
```

Cependant, pour déterminer la clé, il faut conaitre le chiffré ainsi que le clair. Le fichier input.bin possède une information intéressante : le fichier déchiffré s'appelle `congratulations.tar.bz2`. Or, un fichier `tar.bz2` possède un en-tête caractéristique long d'environ 10 octets : `42 5a 68 3X 31 41 59 26 53 59` avec X un chiffre entre 1 et 9. Cet en-tête permettra de ne laisser que $2^{16} * 2^{10} * 9 = 9 * 2^{26} = 603979780$ possibilités à trouver.

Une condition d'arrêt évidente consiste à déchiffrer le fichier `encrypted` et vérifier le hash du résultat.

La recherche par force brute de cet espace en Python est très long, il faut donc réimplémenter l'algorithme en C (annexe E). Malheureusement, même en étant plus rapide, la recherche est toujours trop lente (une semaine nécessaire). Il faut donc améliorer la condition d'arrêt.

Finalement, la consultation de la page Wikipedia sur le format `BZip2` ainsi que la lecture du code source du logiciel `bzip2` [15] permet de trouver des conditions pour améliorer l'efficacité de l'algorithme de recherche. En effet, le format `BZip2` est constitué comme tel :

- 16 bits : `magic` (constant et déjà utilisé pour diminuer la taille de la clé)
- 8 bits : `version` (constant et déjà utilisé pour diminuer la taille de la clé)
- 8 bits : `hundred_k_blocksize` (pris en compte dans l'ensemble des possibilités à essayer)
- 48 bits : `compressed_magic` (constant et déjà utilisé pour diminuer la taille de la clé)
- 32 bits : `crc` (imprédictible)
- 1 bit : `randomize` (la valeur 1 pour ce bit est dépréciée : on peut supposer que le bit est à 0)
- 24 bits : `origPtr` (le logiciel bzip2 vérifie la cohérence de cette donnée)
- 16 bits : `huffman_used_map` (on pourrait supposer que cette valeur vaut `ff ff` comme c'est souvent le cas)

De cette liste, il ressort qu'il est plus facile d'éliminer certains `tar.bz2` improbables grace à la vérification du bit `randomize` et de la valeur de `origPtr` qui doit respecter une condition de validité comme le vérifie `bzip2`.

$$\texttt{origPtr} <= \texttt{hundred\_k\_blocksize} * 10 + 100000 * 9$$

15. http://www.bzip.org/1.0.6/bzip2-1.0.6.tar.gz

Avec cette méthode, il n'est nécessaire de déchiffrer l'ensemble du fichier et calculer le hash résultant que sur un nombre restreint de possibilités. Le champ `huffman_used_map` n'est pas utilisé pour accélérer plus la recherche car certains fichiers `BZip2` récoltés n'ont pas la valeur `ff ff`.

Cet ensemble de méthodes permet de casser le chiffrement en une trentaine de minutes sur un processeur récent. La clé de chiffrément est donc `5ed49b7156fce47de976dac5`.

> Le fichier `tar.bz2` résultant a son champ `hundred_k_blocksize` à 9 comme dans tous les `tar.bz2` générés actuellement. Il avait aussi `huffman_used_map` qui valait `ff ff`. Cet ensemble de conditions supplémentaires aurait raccourci d'autant plus la recherche.

Un fichier `decrypted` est donc obtenu. Il s'agit comme indiqué précédement d'une archive `tar.bz2`. Elle contient le fichier `congratulations.jpg`

```
$ tar xjvf decrypted.tar.bz2
-rw-r--r-- test/test    252569 2015-03-23 10:34 congratulations.jpg
```

Le fichier JPG extrait nous félicite et nous demande un effort pour arriver à la conclusion du challenge.

FIGURE 17 – L'image congratulations.jpg

7 Stéganographie en chaîne

7.1 Stéganographie dans un fichier JPEG

Le fichier `congratulations.jpg` semble à première vue normal. Il s'ouvre parfaitement dans un éditeur d'images. Cependant, il existe plusieurs méthodes pour dissimuler de l'information dans un fichier de ce type.

La stéganographie au sens le plus précis du terme consiste à modifier les pixels de l'image pour y incruster de l'information. Cette technique est détectable en amplifiant les modifications. Pour cela, il est possible d'ajuster la luminosité et le constrate de l'image pour y voir des pixels colorés apparaître. Ce n'est pas le cas dans cette image.

Une autre méthode consiste à cacher l'information dans la structure même du fichier, par exemple dans les méta-données EXIF ou les commentaires JPEG. Pour y voir un peu plus clair, il est possible d'utiliser la commande `djpeg`.

```
$ djpeg -debug congratulations.jpg >/dev/null
libjpeg-turbo version 1.3.0 (build 20131219)
Copyright (C) 1991-2012 Thomas G. Lane, Guido Vollbeding
Copyright (C) 1999-2006 MIYASAKA Masaru
Copyright (C) 2009 Pierre Ossman for Cendio AB
Copyright (C) 2009-2013 D. R. Commander
Copyright (C) 2009-2011 Nokia Corporation and/or its subsidiary(-ies)
```

```
Emulating The Independent JPEG Group's software, version 8d  15-Jan-2012

Start of Image
JFIF APP0 marker: version 1.01, density 89x89  1
Define Quantization Table 0  precision 0
Define Quantization Table 1  precision 0
Start Of Frame 0xc2: width=636, height=474, components=3
    Component 1: 1hx1v q=0
    Component 2: 1hx1v q=1
    Component 3: 1hx1v q=1
Define Huffman Table 0x00
Define Huffman Table 0x01
Start Of Scan: 3 components
    Component 1: dc=0 ac=0
    Component 2: dc=1 ac=0
    Component 3: dc=1 ac=0
  Ss=0, Se=0, Ah=0, Al=1
...
End Of Image
```

Le résultat de la commande montre qu'il s'agit d'un fichier JPEG simple sans ajout d'aucune sorte. Une recherche sur Internet montre qu'il est possible de cacher des données à la fin des données JPEG car les logiciels s'arrêtent au marqueur de fin d'image.

Le marqueur de fin d'image d'un fichier JPEG est composé de la séquence d'octets `0xFF 0xD9`. Une recherche du motif dans le fichier à l'aide d'un éditeur hexadécimal trouve un résultat à l'adresse `0xD7CE`. Le marqueur est immédiatement suivi d'une suite d'octets reconnaissable comme étant le début d'un fichier compressé avec `BZip2`.

```
D7C0| 62 C5 1A 9C  39 8A 82 76  5C 21 03 5F  FE EB FF D9| b...9..v\!._....
D7D0| 42 5A 68 39  31 41 59 26  53 59 BE EC  B4 D2 00 92| BZh91AY&SY......
D7E0| 4B FF FF FF  FF FF FF FF  FF FF FF FF  FF FF FF FF| K...............
D7F0| FF FF FF FF  FF FF FF FF  FF FF FF FF  FF FF FF FF| ................
```

Une fois le fichier compressé isolé, il s'avère qu'il s'agit d'une archive `.tar.bz2`.

```
$ tar xjvf hidden_in_jpg.tar.bz2
-rw-r--r-- test/test    197557 2015-03-23 10:34 congratulations.png
```

Le fichier PNG extrait nous félicite et nous demande deux efforts pour arriver à la conclusion du challenge.

FIGURE 18 – L'image congratulations.png

7.2 Stéganographie dans un fichier PNG

Comme pour le fichier JPEG, l'image semble normale. Après diverses manipulations sur la luminosité et le constraste rien n'apparaît. L'information est donc peut-être cachée dans la structure du fichier. Pour le savoir, l'outil `pngcheck` sera utilisé.

```
$ pngcheck -pvtf congratulations.png
File: congratulations.png (197557 bytes)
  chunk IHDR at offset 0x0000c, length 13
    636 x 474 image, 32-bit RGB+alpha, non-interlaced
  chunk bKGD at offset 0x00025, length 6
    red = 0x00ff, green = 0x00ff, blue = 0x00ff
  chunk pHYs at offset 0x00037, length 9: 3543x3543 pixels/meter (90 dpi)
  chunk tIME at offset 0x0004c, length 7: 27 Feb 2015 13:40:19 UTC
  chunk sTic at offset 0x0005f, length 4919:  illegal reserved-bit-set chunk
...
  chunk sTic at offset 0x1f52d, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x20870, length 38:  illegal reserved-bit-set chunk
  chunk IDAT at offset 0x208a2, length 8192
    zlib: deflated, 32K window, maximum compression
...
  chunk IDAT at offset 0x2e8f6, length 6827
  chunk IEND at offset 0x303ad, length 0
ERRORS DETECTED in congratulations.png
```

Le résultat de la commande est plutôt éloquent : des blocs de type `sTic` (*sic*) sont insérées dans le fichier et inconnues de `pngcheck`. Un script Python permet d'extraire ces blocs. Cependant, et même après une vérification de la cohérence des données, celles-ci ne ressemblent à rien de connu.

```
$ file hidden_in_png
hidden_in_png: data
```

Le programme `pngcheck` indique que les blocs `IDAT` sont compressées avec l'algorithme `zlib`. En décompressant les blocs `sTic` de la même manière, un fichier au format `BZip2` apparaît. Le tout est réalisé par le script Python `dumpPNGChunks.py` (annexe F).

```
$ tar xjvf hidden_in_png.tar.bz2
-rw-r--r-- test/test    904520 2015-03-23 10:34 congratulations.tiff
```

Le fichier TIFF extrait nous félicite et nous demande trois efforts pour arriver à la conclusion du challenge...

FIGURE 19 – L'image congratulations.tiff

7.3 Stéganographie dans un fichier TIFF

Cette fois-ci, l'image n'est pas aussi normale que les précédentes. En effet, une fois le constraste et la saturation manipulés, des pixels colorés apparaissent dans les zones unies de l'image.

Le logiciel Gimp peut être utilisé pour enregistrer les pixels bruts de l'image. Cela permet de manipuler seulement les pixels sans s'occuper des sépcificités du format TIFF.

L'extraction des pixels bruts donne les octets suivants :

FIGURE 20 – L'image congratulations.tiff avec ses pixels révélés

```
00000000  00 01 00 00 00 00 00 00  00 01 00 00 00 01 00 00  |................|
00000010  01 00 01 00 00 01 00 00  00 01 00 01 00 00 01 00  |................|
00000020  00 00 00 00 00 00 00 01  01 00 01 00 00 00 01 00  |................|
...
0000c230  ff ff fe ff ff fe ff ff  ff ff ff ff ff fe fe ff  |................|
0000c240  fe ff ff fe ff ff fe ff  ff fe ff ff ff fe ff ff  |................|
0000c250  ff ff fe fe ff ff ff ff  fe ff ff ff ff ff ff ff  |................|
```

Au vu de ces valeurs, il est possible que l'information à extraire soit présente dans chaque bit de poids faible de chaque octet. En effet, les valeurs de couleurs dans les zones unies ne diffèrent que de sur le bit de poids faible.

Un script Python permet d'extraire ces données et donne le résultat suivant :

```
00000000  40 44 a4 52 01 a2 18 24  02 4a 21 14 48 64 a2 40  |@D.R...$.J!.Hd.@|
00000010  40 04 11 2c 20 00 00 a2  80 2d b6 db 6d b6 db 6d  |@.., ....-..m..m|
00000020  b6 db 6d b6 db 6d b6 db  6d b6 db 6d b6 db 6d b6  |..m..m..m..m..m.|
00000030  db 6d b6 db 6d b6 db 6d  b6 db 6d b6 db 6d b6 db  |.m..m..m..m..m..|
00000040  6d b6 db 6d b6 db 6d b6  db 6d b6 db 6d 00 42 01  |m..m..m..m..m.B.|
```

La commande `file` ne donne rien de concret sur ce fichier. Cependant, le motif `b6 db 6d` qui se répète est intéressant, ainsi que sa position. En effet, un fichier `tar.bz2` obtenu précédemment se présente comme suit :

```
00000000  42 5a 68 39 31 41 59 26  53 59 ea 78 ef 94 01 29  |BZh91AY&SY.x...)|
00000010  ce ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
00000020  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
00000030  ff ff ff e2 b6 5e 78 00  00 00 26 e3 6b e5 e2 90  |.....^x...&.k...|
```

Le motif `ff ff` semble se répéter un nombre équivalent de fois. Le motif `...b6 db 6d...` correspond en binaire à la séquence suivante :

```
...10110110 11011011 01101101...
```

Cela correspond à `ff ff` avec un bit à 0 inséré tous les deux bits. De même, l'en-tête `BZip2` se retrouve. Le script Python suivant extrait le fichier `tar.bz2` caché dans les pixels du fichier brut.

```python
import itertools
import zlib

def grouper(iterable, n, fillvalue=None):
    "Collect data into fixed-length chunks or blocks"
    # grouper('ABCDEFG', 3, 'x') --> ABC DEF Gxx
    args = [iter(iterable)] * n
    return itertools.izip_longest(fillvalue=fillvalue, *args)

def main():
        f = file("congratulations_tiff", "rb")
        bits1 = f.read()#117000)
        f.close()

        bits = []
        for b in grouper(bits1, 3, '\0'):
                bits.append(b[0])
                bits.append(b[1])

        result = []
        for b in grouper(bits, 8, '\0'):
                byte = chr(int("".join(["1" if ord(c) & 0x1 != 0 else "0" for c in b]), 2))
                result.append(byte)

        file("hidden_in_tiff_stripped.tar.bz2", "wb").write("".join(result))


if __name__ == "__main__":
        main()
```

FIGURE 21 – Programme qui extrait les bits de poids faible d'un fichier brut en sautant un bit sur trois

```
$ tar xjvf hidden_in_tiff_stripped.tar.bz2
bzip2: (stdin): trailing garbage after EOF ignored
-rw-r--r-- test/test     28755 2015-03-23 10:34 congratulations.gif
```

Le fichier GIF extrait nous félicite et nous demande quatre efforts pour arriver à la conclusion du challenge...

FIGURE 22 – L'image congratulations.gif

7.4 Stéganographie dans un fichier GIF

Le fichier `congratulations.gif` paraît normal. Il s'ouvre parfaitement dans un éditeur d'images et manipuler la luminosité ne change rien. Cependant, comme la plupart des fichiers GIF il possède une palette, c'est-à-dire un tableau de couleurs que chaque pixel référence.

Gimp permet de consulter et manipuler la palette de couleurs. En utilisant une palette alternative, c'est gagné. En effet, l'adresse de résolution du challenge devient enfin visible !

FIGURE 23 – L'image congratulations.gif avec la palette Bears de Gimp

8 Conclusion et remerciements

Ce challenge aura permis de manipuler divers outils plus ou moins connus connus. Il aura, en outre, permis de (re)découvrir une architecture oubliée, le protocole USB HID, la désobfuscation de code JavaScript et diverses méthodes de stéganographie.

L'auteur tient à remercier les créateurs du challenge pour leurs épreuves variées et instructives, ainsi que Trou pour son aide précieuse dans un grand moment de doute, cde pour ses quelques intuitions géniales sur la fin et sa relecture attentive et Guillaume pour sa relecture tout autant attentive.

A Script du stage 1 : `derubber.py`

```
import functools
import struct

INSTRUCTION = struct.Struct('>BB')

KEYS = [
        "RSVD", "ErrorRollover", "POSTFail", "ErrorUndefined", "a", "b", "c", "d", "e", "f", "g", "h", "i", "j",
        "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", "1", "2",
        "3", "4", "5", "6", "7", "8", "9", "0", "Return", "Escape", "Backspace", "Tab", " ", "-", "=", "[",
        "]", "\\", "#", ";", "'", "`", ",", ".", "/", "CapsLck", "F1", "F2", "F3", "F4", "F5", "F6"
        ]

def main():
    input_file = file('inject.bin')

    lastDelay = 0
    lastBuffer = ""
    for instruction in iter(functools.partial(input_file.read, 2), ''):
        instruction = INSTRUCTION.unpack(instruction)
        if instruction[0] != 0:
            if lastDelay > 0:
                #print "DELAY", lastDelay
                lastDelay = 0

            if instruction[0] == 40:
                if lastBuffer != "":
                    print lastBuffer
                lastBuffer = ""
            else:
                key = KEYS[instruction[0]]
                if len(key) > 1:
                    if lastBuffer != "":
                        print lastBuffer
                    print key
                    lastBuffer = ""
                    continue
                else:
                    if instruction[1] == 1:
```

```
                        if lastBuffer != "":
                            print lastBuffer
                        print "Ctrl+", key
                        lastBuffer = ""
                        continue
                    elif instruction[1] == 8:
                        if lastBuffer != "":
                            print lastBuffer
                        print "Win+", key
                        lastBuffer = ""
                        continue
                    elif instruction[1] == 0:
                        pass
                    elif instruction[1] == 2:
                        key = key.upper()
                    else:
                        raise Exception("Invalid modifier {:x}".format(instruction[1]))
                    lastBuffer = lastBuffer + key
            else:
                lastDelay = lastDelay + instruction[1]

if __name__ == '__main__':
    main()
```

B Script du stage 3 : `draw_mouse.py`

```
import binascii
import struct
from PIL import Image, ImageDraw

HID_REPORT = struct.Struct(">BbbB")

def main():
        in_file = file("mouse_coords.txt")
        out_img = Image.new("RGB", (4096, 4096), "white")
        draw = ImageDraw.Draw(out_img)

        pos = (2048, 2048)
        draw.point(pos, "black")
        for l in in_file:
                data = binascii.unhexlify(l[:-1])
                data = HID_REPORT.unpack(data)
                print data
                pos = (pos[0] + data[1], pos[1] + data[2])
                if data[0] == 0:
                        draw.point(pos, "black")
                        #pass
                elif data[0] == 1:
                        draw.point(pos, "red")

        del draw
        out_img_f = file("mouse_draw.png", "wb")
        out_img.save(out_img_f, "PNG")

if __name__ == "__main__":
        main()
```

C Script du stage 3 : `decipher.cpp`

```
#include <crypto++/serpent.h>
#include <crypto++/modes.h>
#include <crypto++/filters.h>
#include <crypto++/files.h>
#include <crypto++/hex.h>
#include <string>
#include <cassert>

using namespace CryptoPP;
using namespace std;
```

```
int main() {
        string encodedKey = "66c1ba5e8ca29a8ab6c105a9be9e75fe0ba07997a839ffeae9700b00b7269c8d";
        string encodedIv = "535354494332303135 2d537461676533";
        string key, iv;

        StringSource ssk(encodedKey, true, new HexDecoder(new StringSink(key)));
        StringSource ssv(encodedIv, true, new HexDecoder(new StringSink(iv)));
        assert(key.size() == 32);
        assert(iv.size() == 16);

        CBC_CTS_Mode<Serpent>::Decryption ctsDecryption((byte *) key.data(), key.size(), (byte *)
        ↪  iv.data());

        FileSink *destination = new FileSink("decrypted");
        StreamTransformationFilter* cts_dec = new StreamTransformationFilter(ctsDecryption,
        ↪  destination);
        FileSource source("encrypted", true, cts_dec);

        return 0;
}
```

D Script du stage 5 : `decoder.py`

```
import binascii
import hashlib

def transputer4(inp):
        for i in range(12):
                transputer4.total = (transputer4.total + inp[i]) & 0xff
        return transputer4.total
transputer4.total = 0

def transputer5(inp):
        for i in range(12):
                transputer5.mask = (transputer5.mask ^ inp[i]) & 0xff
        return transputer5.mask
transputer5.mask = 0

def transputer6(inp):
        if transputer6.previous is None:
                tot = 0
                for i in range(12):
                        tot = (tot + inp[i]) & 0xffff
                transputer6.previous = tot

        b = (((transputer6.previous & 0x8000) >> 15) ^ ((transputer6.previous & 0x4000) >> 14)) & 0xffff
        b = b ^ ((transputer6.previous << 1) & 0xffff)
        transputer6.previous = b
        return b & 0xff
transputer6.previous = None

def transputer7(inp):
        tot1 = 0
        tot2 = 0
        for i in range(6):
                tot1 = (tot1 + inp[i]) & 0xff
                tot2 = (tot2 + inp[i+6]) & 0xff
        return tot1 ^ tot2

def transputer8(inp):
        transputer8.history[transputer8.nextI] = list(inp)
        transputer8.nextI = transputer8.nextI + 1
        if transputer8.nextI >= 4:
                transputer8.nextI = 0

        msk = 0
        for j in range(4):
                tot = 0
                for i in range(12):
                        tot = (tot + transputer8.history[j][i]) & 0xff
                msk = msk ^ tot
        return msk
transputer8.history = [[0] * 12] * 4
```

```
transputer8.nextI = 0

def transputer9(inp):
        msk = 0
        for i in range(12):
                msk = (msk ^ (inp[i] << (i&7))) & 0xff
        return msk

def transputer10(inp):
        transputer10.history[transputer10.nextI] = list(inp)
        transputer10.nextI = transputer10.nextI + 1
        if transputer10.nextI >= 4:
                transputer10.nextI = 0

        tot = 0
        for j in range(4):
                tot = (tot + transputer10.history[j][0]) & 0xff

        return transputer10.history[tot & 3][(tot >> 4) % 12]
transputer10.history = [[0] * 12] * 4
transputer10.nextI = 0

#def transputer11a(inp):
#        return inp[0] ^ inp[3] ^ inp[7]

#def transputer11b(inp, int12):
#        return inp[int12 % 12]

#def transputer12a(inp):
#        int12 = transputer12a.previous[1] ^ transputer12a.previous[5] ^ transputer12a.previous[9]
#        transputer12a.previous = list(inp)
#        return int12
#transputer12a.previous = [0] * 12

#def transputer12b(inp, int11):
#        return inp[int11 % 12]

def transputer11(inp):
        int12 = transputer11.previous[1] ^ transputer11.previous[5] ^ transputer11.previous[9]
        transputer11.previous = list(inp)
        return inp[int12 % 12]
transputer11.previous = [0] * 12

def transputer12(inp):
        int11 = inp[0] ^ inp[3] ^ inp[7]
        return inp[int11 % 12]

def transputer1(inp):
        i = transputer4(inp) ^ transputer5(inp) ^ transputer6(inp)
        return i

def transputer2(inp):
        i = transputer7(inp) ^ transputer8(inp) ^ transputer9(inp)
        return i

#def transputer3(inp):
#        int11 = transputer11a(inp)
#        int12 = transputer12a(inp)
#        i = transputer10(inp) ^ transputer11b(inp, int12) ^ transputer12b(inp, int11)
#        return i

def transputer3(inp):
        i = transputer10(inp) ^ transputer11(inp) ^ transputer12(inp)
        return i

def transputer0(key, data):
        key = list(key)
        i = 0
        result = []
        for c in data:
                #print "".join(['\\x' + hex(caca)[2:] for caca in key])
                tmp = ((key[i] * 2 + i) & 0xff) ^ ord(c)
                result.append(chr(tmp))

                msk = transputer1(key) ^ transputer2(key) ^ transputer3(key)
                key[i] = msk
```

```
                i = i + 1
                if i >= 12:
                        i = 0
        return "".join(result)


def test():
        key = [ord(c) for c in "*SSTIC-2015*"]
        data = binascii.unhexlify("1d87c4c4e0ee40383c59447f23798d9fefe74fb82480766e")
        result = transputer0(key, data)
        print repr(result)


def break_begin():
        clear = [c for c in "\x42\x5a\x68\x39\x31\x41\x59\x26\x53\x59"]
        cipher = "\xfe\xf3\x50\xdc\x81\xbc\x97\x27\x89\xac"

        for blk in range(1,10):
                clear[3] = chr(0x30 + blk)
                key = []
                for i in range(10):
                        b = ord(clear[i]) ^ ord(cipher[i])
                        if b < i:
                                b = b + 0x100
                        b = b - i
                        b = b / 2
                        print hex(b), chr(b), hex(b | 0x80), chr(b | 0x80)
                        key.append(b)
                return key

DATA = file("encrypted", "rb").read()
HASH = binascii.unhexlify("9128135129d2be652809f5a1d337211affad91ed5827474bf9bd7e285ecef321")
def break_brute():
        key = break_begin() + [0, 0]

        for i in range(256):
                for j in range(256):
                        print i, j
                        key[10] = i
                        key[11] = j
                        plain = transputer0(key, DATA)
                        md = hashlib.sha256()
                        md.update(plain)
                        calc_hash = md.digest()
                        if True:#calc_hash == HASH:
                                file("decrypted_test", "wb").write(plain)
                                print key
                                return

if __name__ == "__main__":
        test()
        #break_begin()
        #break_brute()
```

E Script du stage 5 : `decoder.c`

```
/**
 * gcc -std=c99  -O3 -o decoder decoder.c -lcrypto
*/

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

#include <openssl/sha.h>

static unsigned char KEY[12];
unsigned char *INPUT;
unsigned char *OUTPUT;

unsigned char transputer4(int reset) {
        static unsigned int total = 0;
```

```
        if (reset) {
                total = 0;
        }
        for (unsigned int i = 0; i < 12; i++) {
                total = (total + KEY[i]) & 0xff;
        }
        return total;
}

unsigned char transputer5(int reset) {
        static unsigned int mask = 0;
        if (reset) {
                mask = 0;
        }
        for (unsigned int i = 0; i < 12; i++) {
                mask = (mask ^ KEY[i]) & 0xff;
        }
        return mask;

}

unsigned char transputer6(int reset) {
        static unsigned int total;
        static unsigned char already = 0;
        if (reset) {
                total = 0;
                already = 0;
        }

        if (!already) {
                total = 0;
                for (unsigned int i = 0; i < 12; i++) {
                        total = (total + KEY[i]) & 0xffff;
                }
                already = 1;
        }
        unsigned int b = ((total & 0x8000) >> 15) ^ ((total & 0x4000) >> 14);
        b = b ^ ((total << 1) & 0xffff);
        total = b;
        return b & 0xff;
}

unsigned char transputer7(int reset) {
        unsigned int tot1 = 0;
        unsigned int tot2 = 0;
        for (unsigned int i = 0; i < 6; i++) {
                tot1 = (tot1 + KEY[i]) & 0xff;
                tot2 = (tot2 + KEY[i+6]) & 0xff;
        }
        return tot1 ^ tot2;
}

unsigned char transputer8(int reset) {
        static unsigned int nextI = 0;
        static unsigned char history[12*4] = { 0 };
        if (reset) {
                nextI = 0;
                memset(history, 0, 12*4);
        }

        memcpy(&history[nextI * 12], KEY, 12);
        nextI++;
        if (nextI >= 4) {
                nextI = 0;
        }

        unsigned char mask = 0;
        for(unsigned int j = 0; j < 4; j++) {
                unsigned int total = 0;
                for(unsigned int i = 0; i < 12; i++) {
                        total = (total + history[j * 12 + i]) & 0xff;
                }
                mask = mask ^ total;
        }
        return mask;
}
```

```
unsigned char transputer9(int reset) {
        unsigned int mask = 0;
        for(unsigned int i = 0; i < 12; i++) {
                mask = (mask ^ (KEY[i] << (i&7))) & 0xff;
        }
        return (unsigned char) mask;
}

unsigned char transputer10(int reset) {
        static unsigned int nextI = 0;
        static unsigned char history[12*4] = { 0 };
        if (reset) {
                nextI = 0;
                memset(history, 0, 12*4);
        }

        memcpy(&history[nextI * 12], KEY, 12);
        nextI++;
        if (nextI >= 4) {
                nextI = 0;
        }

        unsigned int total = 0;
        for(unsigned int j = 0; j < 4; j++) {
                total = (total + history[j * 12]) & 0xff;
        }
        return history[(total & 3) * 12 + (total >> 4) % 12];
}

unsigned char transputer11(int reset) {
        static unsigned char previous[12] = { 0 };
        if (reset) {
                memset(previous, 0, 12);
        }
        unsigned int int12 = previous[1] ^ previous[5] ^ previous[9];
        memcpy(previous, KEY, 12);
        return KEY[int12 % 12];
}

unsigned char transputer12(int reset) {
        unsigned int int11 = KEY[0] ^ KEY[3] ^ KEY[7];
        return KEY[int11 % 12];
}

unsigned char transputer1(int reset) {
        return transputer4(reset) ^ transputer5(reset) ^ transputer6(reset);
}
unsigned char transputer2(int reset) {
        return transputer7(reset) ^ transputer8(reset) ^ transputer9(reset);
}
unsigned char transputer3(int reset) {
        return transputer10(reset) ^ transputer11(reset) ^ transputer12(reset);
}

void transputer0(const unsigned char *key, const unsigned char *buffer, unsigned char *obuffer, unsigned int n)
        memcpy(KEY, key, 12);
        unsigned int i = 0;
        unsigned int reset = 1;
        for(; n > 0; n--) {
                *obuffer = ((KEY[i] * 2 + i) & 0xff) ^ *buffer;

                KEY[i] = transputer1(reset) ^ transputer2(reset) ^ transputer3(reset);

                i++;
                if (i >= 12) {
                        i = 0;
                }
                buffer++;
                obuffer++;
                reset = 0;
        }
}

int break_begin(unsigned int blockSize) {
        unsigned char clear[] = "\x42\x5a\x68\x39\x31\x41\x59\x26\x53\x59";
```

```
        const unsigned char *cipher = "\xfe\xf3\x50\xdc\x81\xbc\x97\x27\x89\xac";

        clear[3] = blockSize + '0';

        unsigned char key[12] = { 0 };

        for(unsigned int i = 0; i < 10; i++) {
                unsigned int b = clear[i] ^ cipher[i];
                if (b < i) {
                        b += 0x100;
                }
                b = b - i;
                b = b / 2;
                key[i] = b;
        }

        memcpy(KEY, key, 12);
        return 0;
}

#define MAX_POSS 0x4000000
int brute_force(int argc, char **argv) {
        unsigned int min, max;
        if (argc > 2) {
                unsigned int split, start;
                split = atoi(argv[1]);
                start = atoi(argv[2]);
                min = (MAX_POSS / split) * start;
                max = (MAX_POSS / split) * (start + 1);
                printf("Going from %d to %d excluded\n", min, max);
        }
        else {
                min = 0;
                max = MAX_POSS;
        }

        unsigned char *good_md = "\x91\x28\x13\x51\x29\xd2\xbe\x65\x28\x09\xf5\xa1\xd3\x37\x21\x1a\xff\xad\x91\x
        FILE *f = fopen("encrypted", "rb");
        if (!f)
                return 1;
        fseek(f, 0, SEEK_END);
        long int size = ftell(f);
        rewind(f);

        unsigned char *INPUT = malloc(size);
        unsigned char *OUTPUT = malloc(size);

        if (!fread(INPUT, size, 1, f))
                return 1;
        fclose(f);

        unsigned char key[12] = { 0 };

        SHA256_CTX sha;
        unsigned char md[32];

        for(unsigned int blockSize = 0; blockSize < 10; blockSize++) {
                break_begin(blockSize);
                memcpy(key, KEY, 12);
                for(unsigned int i = min; i < max; i++) {
                        key[0] = (key[0] & 0x7f) | ((i >> 25) & 0x1) << 7;
                        key[1] = (key[1] & 0x7f) | ((i >> 24) & 0x1) << 7;
                        key[2] = (key[2] & 0x7f) | ((i >> 23) & 0x1) << 7;
                        key[3] = (key[3] & 0x7f) | ((i >> 22) & 0x1) << 7;
                        key[4] = (key[4] & 0x7f) | ((i >> 21) & 0x1) << 7;
                        key[5] = (key[5] & 0x7f) | ((i >> 20) & 0x1) << 7;
                        key[6] = (key[6] & 0x7f) | ((i >> 19) & 0x1) << 7;
                        key[7] = (key[7] & 0x7f) | ((i >> 18) & 0x1) << 7;
                        key[8] = (key[8] & 0x7f) | ((i >> 17) & 0x1) << 7;
                        key[9] = (key[9] & 0x7f) | ((i >> 16) & 0x1) << 7;
                        key[10] = (i >> 8) & 0xff;
                        key[11] = (i & 0xff);
                        if ((i & 0xff) == 0)
                                printf("%d\n", i);

                        transputer0(key, INPUT, OUTPUT, 18);
```

```
                        if (OUTPUT[14] & 0x80) {
                                continue;
                        }
                        int origPtr = ((OUTPUT[14] & 0x7f) << 17) | (OUTPUT[15] << 9) | (OUTPUT[16] << 1) | (OUT
                        if (origPtr > 10 + 100000*9) {
                                continue;
                        }

                        transputer0(key, INPUT, OUTPUT, size);

                        SHA256_Init(&sha);
                        SHA256_Update(&sha, OUTPUT, size);
                        SHA256_Final(md, &sha);

                        if (memcmp(md, good_md, 32) == 0) {
                                printf("GOOD: %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x\n",
                                        key[0], key[1], key[2], key[3],
                                        key[4], key[5], key[6], key[7],
                                        key[8], key[9], key[10], key[11]);
                                FILE *of = fopen("decrypted", "wb");
                                if (!of)
                                        return 1;
                                fwrite(OUTPUT, size, 1, of);
                                fclose(of);
                                return 0;
                        }
                }
        }
        return 0;
}

int test() {
        const unsigned char *key = "*SSTIC-2015*";
        const unsigned char *data = "\x1d\x87\xc4\xc4\xe0\xee\x40\x38\x3c\x59\x44\x7f\x23\x79\x8d\x9f\xef\xe7\x4
        unsigned char out[26];

        transputer0(key, data, out, 24);

        out[24] = '\n';
        out[25] = '\0';

        printf("%s", out);

        transputer0(key, data, out, 24);

        out[24] = '\n';
        out[25] = '\0';

        printf("%s", out);

        return 0;
}

int main(int argc, char **argv) {
        //return test();
        return brute_force(argc, argv);
}
```

F Script du stage 6 : dumpPNGChunks.py

```
import struct
import zlib

def read_header(f):
        header = f.read(8)
        if header != "\x89PNG\r\n\x1A\n":
                raise Exception("Invalid header")

CHUNK_HEADER = struct.Struct(">I4s")
def read_chunk(f):
        data = f.read(8)
        if data == '':
```

```
                return None
        (length, typ) = CHUNK_HEADER.unpack(data)
        print length, typ
        data = f.read(length)
        crc = f.read(4)
        return (length, typ, data, crc)

def main():
        f = file("congratulations.png", "rb")
        read_header(f)
        chnk = read_chunk(f)
        data = ''
        while chnk is not None:
                if chnk[1] == 'sTic':
                        data = data + chnk[2]
                chnk = read_chunk(f)
        f.close()

        data = zlib.decompress(data)

        file("hidden_in_png.tar.bz2", "wb").write(data)

if __name__ == "__main__":
        main()
```