Solution Challenge SSTIC 2015

Mouad Abouhali
Airbus Group Innovations

19 mai 2015

1 Premier niveau : le comencement

Le premier niveau se présente sous forme d'un fichier zip comprenant une image d'un carte MicroSD :

Listing 1 –

```
>unzip -l challenge.zip
Archive:  challenge.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
128000000  2015-03-26 14:51   sdcard.img
---------                     -------
128000000                     1 file
```

Le fichier sdcard.img présente l'empreinte suivante :

Listing 2 –

```
0e505d2d0fd16af85df2b56a1d519404500cb9f10db0ebcb46081132502aa6bf
    sdcard.img
```

L'utilitaire file nous indique que c'est un bootsector, une indication qu'on se garde de côté dans un premier temps sans pour autant se lancer dans une analyse du bootsector (on se sait jamais !).

Listing 3 –

```
>file sdcard.img
sdcard.img: x86 boot sector
```

Le plus simple est de monter cette copie et d'inspecter son contenu :

Listing 4 –

```
> mount -o loop sdcard.img /mnt/sdcard
ls -ali  /mnt/sdcard
...
34253730 mars  26 02:49 inject.bin
...
> sha256sum inject.bin
2c818cd06124b8272d9cea87cfddc91343a22c3b7ed645d33ba8790d0199df64
    inject.bin
```

2 Deuxième niveau : on s'échauffe !

À ce stade, on se retrouve avec le fichier "inject.bin" qui ne présente pas un en-tête de fichier connu ou des chaînes de caractères révèlant des informations sur la nature de ce fichier.

Listing 5 –

```
> file inject.bin
inject.bin: data
```

Une recherche sur Internet avec les mots clé : inject.bin, sdcard, etc nous indique qu'il pourait s'agir d'un "DuckyScript" utilisé par une clé USB "Rubber Ducky".

En effet, cette clé USB permet par le biais de son langage de scripting d'exécuter des commandes (en simulant les frappes clavier) sur un système donné. Le langage utilisé reste trés simple, mais une fois finis ce dernier est encodé avant d'être envoyé sur la clé USB "Rubber Ducky".

En supposant que le fichier "inject.bin" est la forme encodé d'un "DuckyScript", l'idée est de récupérer la version initiale. Pour cela, le script perl "ducky-decode" permet facilement de retrouver le code du script initial : [1].

Listing 6 –

```
> ./ducky-decode.pl -h
Ducky-reverse.pl version 0.16
Usage:  ./ducky-decode.pl  [-h] <-f file>

        [-l]            language, supported US(default), UK
        [-h]            this help message
        [-f]            ducky binary file


> ./ducky-decode.pl -f inject.bin > stage2.bin

> file stage2.bin
stage2.bin: ASCII text, with very long lines
```

En examinant le début du fichier de sortie (stage2.bin), on identifie bien des commandes "DuckyScript" :

Listing 7 –

```
00ff007d
GUIR

DELAY500

ENTER

DELAY1000
cmd
ENTER

DELAY50
```

1. https ://code.google.com/p/ducky-decode/

Sans vraiment se plonger dans la documentation du langage de scripting, on peut déduire des premières lignes que le sript lance un invite de commmandes en éxecutant "cmd". La suite du script est un peu plus obscurcie, la commande "powrershelle est exécutée avec le paramètre "-enc" et une suite de caratères (le contenu ci-dessous a été tronqué pour des raisons de lisbilité) :

Listing 8 –

```
powershell
SPACE
-enc
SPACE
ZgB1AG4AYwB0AGkAbwBuACA...
```

L'aide de la commande powershell nous indique clairement que l'option -enc permet de soummettre à la commande powsershell un contenu encodé en base64 :

Listing 9 –

```
PowerShell[.exe] [-PSConsoleFile <file> | -Version <version>]
    [-NoLogo] [-NoExit] [-Sta] [-NoProfile] [-NonInteractive]
    [-InputFormat {Text | XML}] [-OutputFormat {Text | XML}]
    [-WindowStyle <style>] [-EncodedCommand <Base64EncodedCommand>]
    [-File <filePath> <args>] [-ExecutionPolicy <ExecutionPolicy>]
    [-Command { - | <script-block> [-args <arg-array>]
                  | <string> [<CommandParameters>] } ]
...
-EncodedCommand
    Accepts a base-64-encoded string version of a command. Use this
        parameter
    to submit commands to Windows PowerShell that require complex
        quotation
    marks or curly braces.
...
```

En décodant le contenu base64, on arrive au code suivant qui consiste en :

— une fonction qui permet d'écrire des octets au niveau du fichier stage2.zip :

Listing 10 –

```
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();

}
```

— une fonction qui vérifie que le répertoire courant est "challenge2015sstic", sachant que sstic est l'utilisateur "Windows" courant :

Listing 11 –

```
function check_correct_environment{

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

}
```

— et finalement une fonction qui fait appel "check_correct_environment" et écrit dans le fichier "stage2.zip" si la condition citée ci-dessus est vérifiée.

Listing 12 –

```

if(check_correct_environment)
{
   write_file_bytes([Convert]::FromBase64String('UEsDBAoD....
       WnXw==='));
}

else{
write_file_bytes([Convert]::FromBase64String('
    VAByAHkASABhAHIAZABlAHIA'));
}
```

Le message d'erreur affiché dans le cas ou la condition n'est pas vérifiée est le suivant :

Listing 13 –

```
> echo "VAByAHkASABhAHIAZABlAHIA" | base64 -d
TryHarder
```

Le moyen le plus rapide que j'ai tenté était de créer les conditions nécessaires pour l'éxecution du script powershell à citer un répertoire challenge2015 et compte utilisateur "sstic" sous une VM windows (jetable).

un mini traitement du "DuckyScript" a permis d'avoir un fichier batch windows permettant de simuler l'enchaînement des appels "powershell" :

Listing 14 –

```
powershell -enc ZgB1AG4AY...ABBAEgASQBBACcAKQApADsAfQA=
powershell -enc ZgB1AG4AY...ABBAEgASQBBACcAKQApADsAfQA=
```

À la fin de l'éxecution de ce script on obtient le fichier stage2.zip dont l'empreinte est la suivante :

Listing 15 –

```
> sha256sum stage2.zip
9fa22a38989dfe5c0c812eb99094eb3a90df3ae66876b262f42ba14112b6838c
    stage2.zip
```

Le contenu de cette archive est le suivant :

Listing 16 –

```
> unzip -l stage2.zip
Archive:  stage2.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
   501008  2015-03-25 17:17   encrypted
      320  2015-03-25 17:17   memo.txt
  2998438  2015-03-18 10:22   sstic.pk3
---------                     -------
  3499766                     3 files
```

le fichier "memo.txt" nous révèle des élements concernant l'objectif à atteindre au niveau de cette phase :

Listing 17 –

```
Cipher: AES-OFB
IV: 0x53535449433230313 52d537461676532
Key: Damn... I ALWAYS forget it. Fortunately I found a way to hide
    it into my favorite game !

SHA256: 91
    d0a6f55cce427132fc638b6beecf105c2cb0c817a4b7846ddb04e3132ea945
    - encrypted
SHA256: 845
    f8b000f70597cf55720350454f6f3af3420d8d038bb14ce74d6f4ac5b9187 -
     decrypted
```

À ce niveau, il faut récupérer une clé de chiffrement qui va servir à déchiffrer le le fichier encrypted. Il reste alors le fichier "sstic.pk3", on peut se demander que c'est ce dernier qui v a contenir cette clé de chiffrement.

Le premier lien sur Internet obtenu suite à une rechreche basée sur le mot clé ".pk3" nous permet de situer un peu le contexte de cette phase.

D'après Wikipédia :

Listing 18 –

```
A .PK3 file is a renamed ZIP file, used in games based on the Quake
     III engine.
...The .PK3 file contains the maps, models, textures, weapons,
sounds, scripts and other assets for the game.
...
```

Cela est un bon signe ! car Quake 3 peut correspondre au "favorite game" cité au niveau du mémo. On peut attaquer le problème selon deux manières, soit charger la map "sstic" dans un environement de jeu comme "OpenArena" ou s'amuser à étudier le structure d'un fichier PK3 afin d'élaborer une autre stratégie (comme l'édition de la map, son analyse et sa modification) .

Pour les "gamers", ce stage est purement une partie de plaisir, pour les autres , comme moi, il faut apprendre les bases comme : comment charger une map donné ?

Après avoir lancé "OpenArena", on arrive au menu principal (on s'assure que la map sstic est dans répertoire baseoa d'openArena). Une fois l'environnement lancé, on accède à la console (SHIFT + ESC) et charge la map sstic avec la commande "sstic ".

On arrive directement sur "arena sstic " et on est accueilli avec les message suivant :

L'image listée ci-dessus met en évidence la présence d'image contenant potentiellement des parties de la clé qu'on recherche. La suite du tour nous révèle d'autres images avec cette fois-çi des signes différents et à chaque une suite de caractère en hexadécimal en orange, blanc puis vert :

— image avec le signe "WIFI" :

— image avec le signe "WIFI" :

— image avec le signe "drapeau" :

Et au fur et à mesure qu'on se balade dans l'arène sstic on découvre d'autre images cachées un peu partout :

— image cachée derrière un coffre :

— caché sous un escalier

— caché sur un coffre suspendu :

Ce coffre reste facilement accessible avec saut "Rocket Jump".

— caché derrière un portail :

Ce portail doit être activé depuis un bouton présent ailleurs au niveau de l'arène :

Au fond de l'arène on se retrouve face à un grande porte avec "SSTIC" marqué dessus, l'idée ici est d'ouvrir la porte et d'accéder au contenu se trouvant derrière cette porte. le mode "god" (sstic puis) s'avère trés utile pour ne pas perdre trop de temps :

Une fois passé ce portail, quelques obstacles se présentent notamment celui là :

Une maitrise du saut "Rocket Jump" s'avère nécessaire :

Puis on arrive au bout du chemin où il faut activer l'entrée à une salle secrète en tirant sur le bouton

Et là , on est acceulli avec un message nous indiquant qu'on a bien trouvé la clé :

La clé se présente sous la forme suivante :

Il suffit alors de concatener les différentes parties de chaque image trouvées sur l'arène et en choisissant la bonne couleur. Cela nous donne la clé suivante :

Listing 19 –

```
9E 2F 31 F7
81 53 29 6B
3D 9B 0B A6
76 95 DC 7C
B0 DA F1 52
B5 4C DC 34
FF E0 D3 55
26 60 9F AC
```

Ayant ces élements, on peut exécuter la commande openssl suivante pour récupérer le fichier déchiffré :

Listing 20 –

```
openssl aes-128-ofb -K key -iv iv -e -in encrypted -out decrypted

sha256sum decrypted
f9ca4432afe87cbb1fca914e35ce69708c6bfa360b82bff21503b6723d1cfbf0
    decrypted

file decrypted
decrypted: Zip archive data, at least v1.0 to extract


unzip -l decrypted
Archive:  decrypted
  Length      Date    Time    Name
---------  ---------- -----   ----
   296798  2015-03-25 17:06   encrypted
      330  2015-03-25 17:14   memo.txt
  2347070  2015-03-03 10:14   paint.cap
---------                     -------
  2644198                     3 files
```

3 Troisième niveau : on garde le rythme !

Le fichier memo.txt nous indique que notre objectif est de déchiffrer le fichier encrypted fourni en utilisant l'iv donné et l'algorithme Serpent-1-CBC-With-CTS.

Ce niveau nous présente deux difficultés, la première est biensûr retrouver la clé qui est potentiellement cachée dans le fichier "paint.cap". La deuxième difficulté correspond à l'absence d'un outil (comme openssl) qui nous permetterait de déchiffrer notre fichier encrypted en utilisant l'algorithme indiqué.

Trouvons d'abord la clé. Le fichier "paint.cap" est une capture d'un échange entre un periphérique USB et un ordinateur.

Donc, notre objectif est de comprendre cet échange, identifier sa nature et cela en identifiant la nature de l'équipement qui a ét connecté à notre machine.

Les premiers échanges nous indique qu'il s'agit d'une souris :

Listing 21 –

```
...
Frame 2: 82 bytes on wire (656 bits), 82 bytes captured (656 bits)
USB URB
DEVICE DESCRIPTOR
    bLength: 18
    bDescriptorType: DEVICE (1)
    bcdUSB: 0x0200
    bDeviceClass: Device (0x00)
    bDeviceSubClass: 0
    bDeviceProtocol: 0 (Use class code info from Interface
        Descriptors)
    bMaxPacketSize0: 8
    idVendor: IBM Corp. (0x04b3)
    idProduct: Wheel Mouse (0x310c)
    bcdDevice: 0x0200
    iManufacturer: 0
    iProduct: 2
    iSerialNumber: 0
    bNumConfigurations: 1
...
```

3.1 Le champ LeftOver Capture Data

En effet en étudiant un peu la documentation concernant les échanges USB d'un souris, on arrive à identifier que le champ "LeftOver Capture Data" contient les coordonnées de la souris à chaque mouvmenent [2] .

3.2 Extraction des données

À ce niveau, il ne reste qu'à extraire ces coordonnées et essayer de reproduire les mouvements de la souris. Idéalement tracer ces mouvements afin d'avoir une image.

L'extraction peut être réalisée avec la ligne tshark suivante :

Listing 22 –

```
tshark.exe -r "paint.pcap" -T fields -e usb.capdata -R >> usbdata
```

2. http ://wiki.osdev.org/Mouse_Input

Cela nous permet d'avoir un fichier contenant le contenu du champ LeftOver Capture Data de l'ensemble des paquets :

Listing 23 –

```
...
00:fe:00:00
00:ff:00:00
00:fe:00:00
00:ff:00:00
00:fe:00:00
00:fe:00:00
00:fe:00:00
00:fe:00:00
00:fd:00:00
00:fd:ff:00
...
```

D'après le lien cité précédemment on arrive à comprendre la structure des doonées envoyés par la souris :

Listing 24 –

```
...
Even if your mouse is sending 4 byte packets, the first 3 bytes
    always have the same format. The first byte has a bunch of bit
    flags. The second byte is the "delta X" value -- that is, it
    measures horizontal mouse movement, with left being negative.
    The third byte is "delta Y", with down (toward the user) being
    negative. Typical values for deltaX and deltaY are one or two
    for slow movement, and perhaps 20 for very fast movement.
    Maximum possible values are +255 to -256 (they are 9-bit
    quantities, two's complement).
```

On en conclut que le deuxième octet correspond à la valeur de déplacement horizontal de la souris (delta x) et le troisième octet correspond à la valeur de déplacement vertical de la souris.

En ayant ces coordonnées, l'idée maintenant est de produire un fichier de coordonnées (x,y) qu'on peut fournir à gnuplot afin qu'il nous dessine les mouvements de la souris.

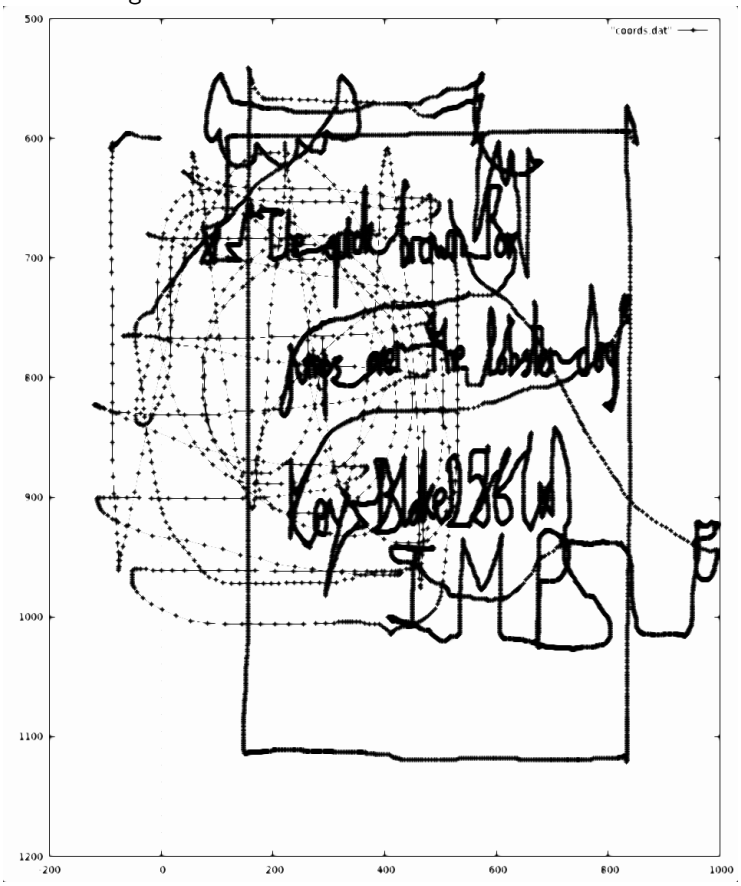
On négligera pour le moment les octets permettant de savoir si le boutons de la souris sont activés. La description du format du champ "LeftOver Capture data" nous permet d'écrire le script python suivant :

Listing 25 –

```
import struct

fd  = open("usb_data000","r")
lines = fd.readlines()
fd.close()

lines_test = lines[:10]
list_coord = []
x=0
y=0
d_x = 0
d_y = 600
temp=""

for line in lines:
```

```
16        ...
17        x =  struct.unpack('b',x)[0] + d_x
18        ...
19        y =  struct.unpack('b',y)[0] + d_y
20        ...
21        list_coord.append((x,y))
22        d_x = x
23        d_y = y
24
25  fplot  =  open("coords.plot","w")
26  temp = ""
27  for i in list_coord:
28        print(i)
29        x,y = i
30        temp = str(x)+"     "+str(y)+"\n"
31        print(temp)
32        fplot.write(temp)
33
34
35  fplot.close()
```

Le fichier produit par ce script, peut être chargé dans gnuplot et nous permet d'afficher l'image suivante :



En lisant le contenu de l'image, on s'aperçoit que la clé est le hash de la phrase suivante en utilisant l'algorithme Blake256. Une implémentation en C est facilement

téléchargeable sur Internet à cet URL [3]

Listing 26 –

```
The quick brown fox jumps over the lobster dog
```

Listing 27 –

```
> blake256 key2.txt
66c1ba5e8ca29a8ab6c105a9be9e75fe0ba07997a839ffeae9700b00b7269c8d
    key2.txt
```

3.3 Cryptopp et Serpent

Une fois la clé identifiée, il ne reste plus qu'à écrire le code nécessaire pour l'utilisation de l'ensemble des éléments dont nous disposons. Cryptopp est une bibliothèque en C++ qui offre l'implémentation de plusieurs algorithme de cryptographie notamment Serpent. L'intérêt de cette librairie est la possibilité d'utiliser CBC avec CTS pour le le déchiffrement Serpent.

Ci-dessous un extrait du code utilisé (cf. serpent_cbc_cts.c pour la totalité du code)

Listing 28 –

```
...
int main(int argc, char* argv[])
{
AutoSeededRandomPool prng;

 bool pass=true, fail;

   byte key[32] = { 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] = {0x53,0x53,0x54,0x49,0x43,0
            x32,0x30,0x31,0x35,0x2d,0x53,0x74,0x61,0x67,0x65,0x33};

        string plain = "This is my clear text to give to the
            Seprent";
        string cipher, encoded, recovered;



        // Opening encrypted file
         std::ifstream  infile ("encrypted", std::ios::binary);

        cout << " [+] Opening encrypted file OK " << endl;
       //Opening decrypted file
        std::ofstream outfile ("decrypted", std::ios::binary);
        cout << " [+] Opening decrypted file OK " << endl;


        /*********************************\
        \*********************************/

```

3. https ://131002.net/blake/

```
        // Pretty print key
        encoded.clear();
        StringSource ss1(key, sizeof(key), true,
                new HexEncoder(
                        new StringSink(encoded)
                ) // HexEncoder
        ); // StringSource
        cout << "key: " << encoded << endl;

        // Pretty print iv
        encoded.clear();
        StringSource ss2(iv, sizeof(iv), true,
                new HexEncoder(
                        new StringSink(encoded)
                ) // HexEncoder
        ); // StringSource
        cout << "iv: " << encoded << endl;

        /*********************************\
        \*********************************/

        try
        {
                cout << "plain text text text :" << plain << endl;
                CBC_CTS_Mode< Serpent >::Encryption e;
                e.SetKeyWithIV(key, sizeof(key), iv);
                StringSource ss3(plain, true,
                        new StreamTransformationFilter(e,
                                new StringSink(cipher)
                        ) // StreamTransformationFilter
                ); // StringSource

           }
         catch(const CryptoPP::Exception& e)
         {
                 cerr << e.what() << endl;
                 exit(1);
             }

        try
       {
                  CBC_CTS_Mode< Serpent >::Decryption d;
                 d.SetKeyWithIV(key, sizeof(key), iv);
                CryptoPP::FileSource (infile, true,
                        new StreamTransformationFilter(d,
                                        new CryptoPP::FileSink(
                                            outfile)
                        ) // StreamTransformationFilter
                         ); // Filesource;*/
               }
         catch(const CryptoPP::Exception& e)
         {
                 cerr << e.what() << endl;
                  exit(1);
         }

...
```

Le code produit à la fin le fichier decrypted suivant :

Listing 29 –

```
> file decrypted
```

```
decrypted: Zip archive data, at least v2.0 to extract

> unzip -l decrypted
Archive:  decrypted
  Length      Date    Time    Name
---------  ---------- -----   ----
   540003  2015-03-25 16:28   stage4.html
---------                     -------
   540003                     1 file

> sha256sum decrypted
7beabe40888fbbf3f8ff8f4ee826bb371c596dd0cebe0796d2dae9f9868dd2d2
```

4 Quatrième niveau :le mur !

Le niveau 4 se présente sous la forme d'un fichier html (stage4.html) contenant un script JavaScript obscurci.

Listing 30 –

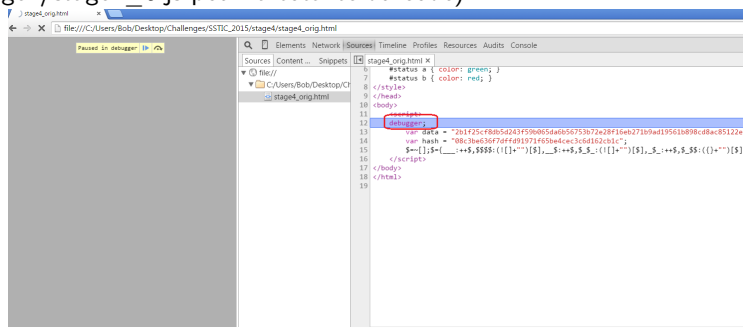
```
...

 var data = "2b1f25..."
 var hash = "08c3be..."
....
 $=~[];$={___:++$,$$$$:(![]+"")[$],__$:++$,$_$_:(![]+"")[$],_$_:++$
      ,$_$$:({}+"")[$],$$_$:($[$]     +"")[$],_$$:++$,$$$_:(!""+"")[$
      ],$__:++$,$_$:++$,$$__:({}+"")[$],$$_:++$,$$$:++$,$___:++$,$__
      $:++$};$.$_=($.$     _=$+"")[$.$_$]+($._$=$.$_[$.__$])+($.$$=($
      .$+"")[$.__$])+((!$)+"")[$._$$]+($.__=$.$_[$.$$_])+($.$=(!""+"
      ")[$.     __$])+($._=(!""+"")[$._$_])+$.$_[$.$_$]+$.__+$._$+$.$
      ;$.$$=$.$+(!""+"")[$._$$]+$.__+$._+$.$+$.$$;$.$=($.___)[     $.
      $_][$.$_];$.$($.$($.$$+"\""+"__="+$.$$_$+$._$+$.$$__+$._+"\\"+
      $.__$+$.$_$+$.$_$+$.$$$_+"\\"+$.__$+$.$_$+$.$     $_+$.__+";
...
```

La première étape va consister à rendre le code JavaScript plus lisible afin de comprendre son déroulement.

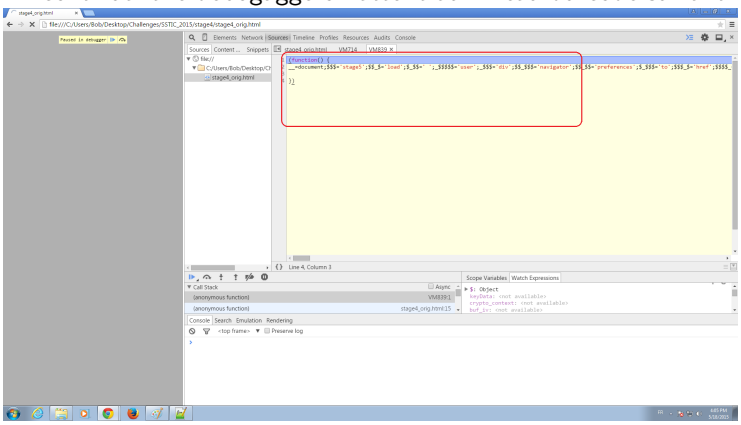
Pour cela on peut avoir recour au debugger JavaScript De chrome, qui va nous permettre de récupérer le contenu JavaScript au fur et à mesure de son execution.

Pour cela on ajoute la clause "debugger() ;" au niveau du code JavaScript et on execute pas à pas ce dernier et on choisissant de "sauter" à l'appel de chaque de fonction.

Avec cette technique on arrive à construire un code qui ressemble à ce-ci (cf. stage4/stage4_0.js pour la totalité du code) :



En continuant le débogagge on attérit au niveau du code suivant :



On arrive après ce premier niveau d'obfuscation au code suivant :

Listing 31 –

```
function () {
  __ = document;
  $$$ = 'stage5';
  $$_$ = 'load';
  $_$$ = ' ';
  _$$$$$ = 'user';
  _$$$ = 'div';
  $$_$$$ = 'navigator';
  $$_$$ = 'preferences';
  $_$$$ = 'to';
  $$$_$ = 'href';
  $$$$_ = '=';
  $$$$$ = 'chrome';
  _$$$$ = '"';
  $_$$$$ = 'Agent';
  $$$_$$ = 'down';
  $$$$_$ = 'import';
  $ = '<b>Failed' + $_$$ + $_$$$ + $_$$ + $$_$ + $_$$ + $$$ + '</b>
      ';
  ___ = 'write';
  ____ = 'getElementById';
  _$_ = "raw";
  $$ = 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 (___[____________] < _____________)
        ______ += _______;
      ______ += ___;
    }
```

```
89      return ______;
90    }
91    function ____________________________() {
92      $_ = _____(__________[______________](__________[
            ___________________](______$_) + ______________,
            ___________________));
93      _$__ = _____(__________[______________](__________[
            ___________________](_______$) - ___________________,
            ___________________));
94      _$ = {};
95      _$[_$______] = __$_____;
96      _$[___$____] = $_;
97      _$[____________] = _$__[____________] * _________________;
98      __$[$____](_$_, _$__, _$, false, [__$_])[__$__](function (_$___
            ) {
99        __$[__$_](_$, _$___, ______________________(____$_))[__$__](
              function (___$_) {
100         ____$ = new ________________________(___$_);
101         __$[_$____](___$__, ____$)[__$__](function (____$$) {
102           if (_____$ == ________________________(new
                ________________________(____$$))) {
103             _____$_ = {};
104             _____$_[______$] = $_______;
105             _____$ = new _$_____([____$], _____$_);
106             __$____ = ___$___[____$__](_____$);
107             __[____](___$)[__$___] = ____$___ + __$____ + _____$__;
108           } else {
109             __[____](___$)[__$___] = $;
110           }
111         });
112       }).catch (function () {
113         __[____](___$)[__$___] = $;
114       });
115     }).catch (function () {
116       __[____](___$)[__$___] = $;
117     });
118   }
119   $$[$________](____________________________, $_$);
120
121 }
```

En utilisant des outils accessibles sur Internet (http ://esprima.org/demo/rename.html) on peut indenter le code, et puis renommer les variables avec des noms qui nous parlent en fonction du contexte du code.

Après un travail essentiellement manuel, on arrive à un code qui ressemble à ce ceci (cf. stage4/stage4.js pour la totalité du code) :

Listing 32 –

```
1  str_document = document;
       str_stage5 = 'stage5';
2   str_load = 'load';
3   str_space = ' ';
4   str_user = 'user';
5   str_div = 'div';
6   str_navigator = 'navigator';
7   str_preference = 'preferences';
8   str_to = 'to';
9   str_href = 'href';
10 ...
11 function f4() {
```

```
12          // a l'origine keydata and buf_iv = f1(useragent.substr(
                indeof(()))
13                  //tring chaine.substr(Integer debut, Integer
                        longueur)
14                 //useragent.substr(user_agent.strindexof('(') +1)
                       ,16 )
15          buf_iv = f1(str_window_user_agent[str_substr](
                str_window_user_agent[str_index_of](str_accolade_ouv
16  rante) + int_int_val_zero_plus_un, int_64));
17          //user_agent.substr( useragent.indexof(')') -16  ,16)
18                  keyData = f1(str_window_user_agent[str_substr](
                        str_window_user_agent[str_index_of](str_acc
19  olade_fermante) - int_64, int_64));
20          crypto_context = {};
21          crypto_context[str_name] = str_AES_CBC;
22          crypto_context[str_iv] = buf_iv;
23          crypto_context[str_length] = keyData[str_length] *
                int_int_val_zero_plus_un_mul_2_mul_quatre;
24                  //var result = crypto.importKey(format, keyData,
                        algo, extractable, usages);
25          str_crypto_subtle[str_import_Key](str_raw, keyData,
                crypto_context, false, [str_decrypt])[str_then
26  ]
27  ...
```

Cela va nous permettre de mieux comprendre le code. L'essentiel du code se concentre au niveau des lignes suivantes :

Listing 33 –

```
1  ...
2  crypto_context = {};
3  crypto_context[str_name] = str_AES_CBC;
4  crypto_context[str_iv] = buf_iv;
5  crypto_context[str_length] = keyData[str_length] *
       int_int_val_zero_plus_un_mul_2_mul_quatre;
6  //var result = crypto.importKey(format, keyData, algo, extractable,
       usages);
7
8
9   //var result = crypto.decrypt(algo, key, ciphertext);
10   str_crypto_subtle[str_decrypt](crypto_context, _$___, f2(buf_data)
       )[str_then](function(___$_)
11 ...
```

En effet, ce code JavaScript basé sur la librairie JavaScript "Subtle" déchiffre le contenu de la variable "buf_data" en utilisation l'algorithme AES-CBC. sachant que la clé et l'iv sont générée par le biais de deux fonctions.

En déboguant une deuxième fois, on se rend rapidement compte que ces deux fonctions prennent en paramètre le User-Agent de l'explorateur Internet utilisé et lui applique un certain traitement qu'on résumera par les deux lignes python suivante :

Pour l'iv :

Listing 34 –

```
1  user_agent_iv = user_agent[user_agent.index('(')+1:user_agent.index
       ('(')+1 + 16 ]
```

Pour la clé

Listing 35 –

```
user_agent_key =  user_agent[user_agent.index(')')-16:user_agent.
    index(')')]
```

Une fois le principe de ces deux fonctions compris, on peut le traduire en python comme ceci (cf stage4/f1.py pour la totalité du code) :

Listing 36 –

```
...
fi = open("iv.txt","w")
fk =  open("keys.txt","w")


for ua in list_user_agents :
        user_agent =  ua.strip()
        print("[+] Analyzing : %s : \n")%user_agent
        user_agent_iv = user_agent[user_agent.index('(')+1:
            user_agent.index('(')+1 + 16 ]
        result=""
        iv = list()
        for i in user_agent_iv:
            iv.append(hex(ord(i)).lstrip('0x'))
        result  = ''.join(str(x) for x in iv)
        fi.write(result+"\n")

        user_agent_key =  user_agent[user_agent.index(')')-16:
            user_agent.index(')')]
        key  = list()
        result=""
        for i in user_agent_key:
            key.append(hex(ord(i)).lstrip('0x'))
            data = struct.pack('B', ord(i))
        result = ''.join(str(x) for x in key)

G        fk.write(result+"\n")


fi.close()
fk.close()
```

En analysant un peu le début du code, on se rend compte qu'une vérification de la présence de la ressources "preferecences.xul" est effectuée. On sait que cette ressources n'est accessible que le sous l'explorateur Internet FireFox. On peu supposer alors que le code actuel génère une clé et un iv qu'à partir d'un User-Agent de type Firefox.

La structure d'un User-Agent FireFox est décrite sur le site : https ://developer.mozilla.org/en-US/docs/Web/HTTP/Gecko_user_agent_string_reference.

Globalement, la structure d'un User-Agent FireFox est de la forma suivante :

Listing 37 –

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

Pour la génération des User-Agent, on peut se servir du script python qui a pour objectif de nous produire l'ensemble des User-Agent propre à FireFox et toute platforme confondue :

Listing 38 –

```

```

```
2  fd = open("ALL.txt","w")
3
4  platform_win = ["Windows NT %d.%d;"%(5,y) for y in range (0,10) ]
5  platform_win += ["Windows NT %d.%d;"%(6,y) for y in range (0,10) ]
6  platform = ["%s Win64; x64;"%(p) for p in platform_win ]
7  platform += ["%s WOW64;"%(p) for p in platform_win]
8  platform_mac = ["Macintosh; Intel Mac OS X %d.%d;"%(10,y) for y in
       range (0,15) ]
9  platform_mac += ["Macintosh; PPC Mac OS X %d.%d;"%(10,y) for y in
       range (0,15) ]
10 platform += platform_mac
11 platform_lin = ["X11; Linux i686;", "X11; Linux x86_64;" ,"X11;
       Linux i686 on x86_64;", "Maemo; Linux armv
12 7l;"]
13 platform += platform_lin
14 platform_andro = ["Android; Mobile;","Android; Tablet;" ,"Mobile;"
       , "Tablet;"]
15 platform += platform_andro
16 rv = ["rv:%d.0"%(x) for x in range(0,40)]
17 for p in platform:
18     for v in rv:
19         s  = "(%s %s)"%(p,v)
20         print(s)
21         fd.write(s+"\n")
22
23
24 fd.close()
```

Après avoir généré l'ensemble des User-Agent possibles, puis l'ensemble des clés et des IV, il ne reste qu'à passer le tout au script suivant qui va nous automatiser le déchiffrement.

Listing 39 –

```
1 ...
2 x  = 0
3 with open("iv.txt") as fi , open("keys.txt") as fk :
4     for i,k in itertools.izip(fi,fk):
5         ...
6         os.system(openssl_command)
```

Ci-dessus l'empreinte du chiffré utilisé :

Listing 40 –

```
1
2 > sha256sum data.bin
3 3ffadce0ae59a0c34437d9c85c82ad41456587bb3a749317e27d76edb7e799da
      data.bin
```

Pour la vérification du résultat, on sait que le code JavaScript réalise une vérification de l'empreinte du fichier déchiffré avec la valeur :

Listing 41 –

```
1
2 08c3be636f7dffd91971f65be4cec3c6d162cb1c
3
4 \begin{lstlisting}
5
6 > shasum  * | grep 08c3be636f7dffd91971f65be4cec3c6d162cb1c
7 08c3be636f7dffd91971f65be4cec3c6d162cb1c  data.dec1875
8
```

```
9
10 > file data.dec1875
11 data.dec1875: Zip archive data, at least v2.0 to extract
12
13 > unzip -l data.dec1875
14 Archive:  data.dec1875
15   Length      Date    Time    Name
16 ---------  ---------- -----   ----
17    253083  2015-03-24 17:15   input.bin
18     13089  2015-03-25 13:19   schematic.pdf
19 ---------                     -------
20    266172                     2 files
```

5 Les transputers !

On arrive au niveau 5 avec le fichier zip précédant qui contient un fichier pdf décrivant l'architecture globale du code fourni au niveau du fichier input.bin.

D'après le message fourni pour avec le vecteur de test :

Listing 42 –

```
1
2 decrypt(key, data) ==  I  love ST20 architecture
```

On se doute bien qu'on est face à code d'une architecture ST20. Pour cela, on procède par une analyse statique. Certes il existe un émulateur st20 mais je ne vous cache pas que je n'ai pas trop regardé son fonctionnement.

Donc, armé d'IDA pro et de "l'instruction set ST20" j'attaque l'analyse statique du code.

Ma démarche a consisté en premier à identifier les différents transputers en me servant du fichier schematics.pdf comme guide. Mais avant cela je suis parti examiner le début du code désassemblé par IDA. Mon fil conducteur est la liaison établie entre les transputers par le moyen des "channels".

La spécification ST-20 nous permet de mettre en évidence le mécanisme des "channels" :

Listing 43 –

```
1
2 A channel is used for synchronization and data-transfer between two
      processes. It may be
3 implemented by:
4 + a word in memory - for communication between processes on a
     single ST20-C2 (internal
5 channel),
6 + an external link - for communication between:
7  - two ST20- C 2 s  or
8  - an ST20-C2 and an external device (via an external channel) or
      an internal
9 subsystem.
```

Selon la spécification ST-20 les channels sont découpés de la manière suivante :

Listing 44 –

```
1
2 #80000020 Event Channel
3 #8000001C External (Boot) Channel In 3
```

```
#80000018 External (Boot) Channel In 2
#80000014 External (Boot) Channel In 1
#80000010 External (Boot) Channel In 0
#8000000C External Channel Out 3
#80000008 External Channel Out 2
#80000004 External Channel Out 1
#80000000 External Channel Out 0
```

Il est à noter qu'un numéro de channel prend des valeurs de 0x8000 0000 , 0x80000 0004 en écriture, et des valeurs allant de 0x8000 0010 , 0x80000 00014 en écriture.

Les deux fonctions prennent en paramètres la taille de la données à lire ou à écrire, le numéro de channel et un pointeur vers le buffer de la donnée à lire ou à écrire. Ci-desssous un exemple d'écriture :

Listing 45 –

```

out                      ; Channel 0 :Ecriture
                         ;
                        ; Out(len= 8,channel = 0x8000000 , message =
                            0x7ff500dd
```

Listing 46 –

```

in                       ; Channel 0 :Lecture
                         ; in(len= 0x4,channel = 0x80000010 ,
                            message = L2 )
                         ;
```

Ces deux instructions vont me permettre de suivre l'éxecution du code, en effet la première partie du code (supposée être le transputer0) se chargent de distribuer les différentes portions du code au différents trasnputer :

On découvre en analysant le processus de distribution de code, que chaque partie de code d'un transputer se termine par une structure de la forme suivante :

Listing 47 –

```


ROM:7FF500DB ; End of function Transpute0
ROM:7FF500DB
ROM:7FF500DB ; ---------------------------------------------------
ROM:7FF500DD aBootOk:         db "Boot ok",0
ROM:7FF500E5 aCodeOk:         db "Code Ok",0
ROM:7FF500ED aDecrypt:        db "Decrypt",0
ROM:7FF500F5                  dd 0F022BC24h
ROM:7FF500F9                  dw 71h
ROM:7FF500FB                  dw 0
ROM:7FF500FD                  dd 80000004h            ; channel 1
    --> pour le code du transputer1
ROM:7FF50101                  dd 0
ROM:7FF50105
ROM:7FF50105 ; =============== S U B R O U T I N E
    =======================================
```

Listing 48 –

```

ROM:7FF50170 ; End of function Transpute1
```

```
ROM:7FF50170
ROM:7FF50170 ; ---------------------------------------
ROM:7FF50172                 dd 0F022B800h
ROM:7FF50176                 dw 71h
ROM:7FF50178                 dw 0
ROM:7FF5017A                 dd 80000008h            ; channel 2
    --> pour le code du transputer2
ROM:7FF5017E                 dd 0
ROM:7FF50182
ROM:7FF50182 ; =============== S U B R O U T I N E
    =======================================
```

Cette struture est utilisée pour connaitre la taille du code correspondant à chaque transputer (ex : 0x71) ainsi que le numéro du channel du transputer.

Le message "Boot OK" est renvoyé par le transputer0, indiquant le traitement va commencer .

Listing 49 –

```
....
ldc     0C9h ; ''
ldpi                    ;  Areg <- 0x7ff500dd
mint
ldc     8
out                     ; Channel 0 :Ecriture
                        ;
                        ; Out(len= 8,channel = 0x8000000 , message
                            = 0x7ff500dd)
...
ROM:7FF500DD aBootOk:        db "Boot ok",0
...
```

Les 4 premiers transputers se chargent en premier lieu de distribuer le code des transputers. Et cela en lisant 12 octets à chaque fois, vérifiant que les 4 premiers octets lus ne sont pas nuls :

Listing 50 –

```

ROM:7FF50018                 ldlp    49h ; 'I'
ROM:7FF5001A                 mint
ROM:7FF5001C                 ldnlp   4
ROM:7FF5001D                 ldc     0Ch
ROM:7FF5001E                 in                      ; Channel 0 :
    lecture
ROM:7FF5001E                                         ; in(len= 0xC,
    channel = 0x80000010 , message = WPTR @ 0x49(*4) )
ROM:7FF5001F                 ldl     49h ; 'I'
ROM:7FF50021                 cj      loc_7FF50038    ; Jump if Areg
     is 0
ROM:7FF50023                 ldc     0CDh ; ''
ROM:7FF50025                 ldpi                    ;  Areg <- 0
    x7ff500f4
ROM:7FF50027                 mint
ROM:7FF50029                 ldnlp   4
ROM:7FF5002A                 ldl     49h ; 'I'
ROM:7FF5002C                 in                      ; Channel 0 :
    lecture
ROM:7FF5002C                                         ; In(len= word
    [WPTR @ 0x49(*4)],channel = 0x8000010 , message =  0x7FF500F4 )
ROM:7FF5002D                 ldc     0C3h ; ''
ROM:7FF5002F                 ldpi                    ;  Areg <- 0
    x7ff500f4
```

```
19 ROM:7FF50031                  ldl     4Ah ; 'J'
20 ROM:7FF50033                  ldl     49h ; 'I'
21 ROM:7FF50035                  out                     ; Out(len=
       word[WPTR @ 0x49(*4)],channel = word[WPTR @ 0x4A(*4)] , message
        =0x7ff500
```

Une fois le code chargé au niveau des différents transputer, un message "Code OK" est retourné par le transputer0 :

Listing 51 –

```
1  ...
2  ldc     94h ; ''
3  ldpi                    ;  Areg <- 0x7ff500e5
4  mint
5  ldc     8
6  out                     ; Channel 0
7                          ;
8                          ; Out(len= 8,channel = 0x8000000 , message
                               = 0x7ff500e5
9  ...
10 ROM:7FF500E5 aCodeOk:       db "Code Ok",0
11
12 ...
```

Une fois qu'on a identifié cette partie (distribution de code) , on peut se concentrer sur le traitement propore à chaque transputer.

Pour la lecture du code, il faut s'habituer à quelques spécificités (cette liste reste non exhausuitive) ST-20 telles que :

les instructions agissent essentiellement sur trois registres Areg, Breg et Creg (et WPTR) ;

certaines instructions font décaler les trois registre à l'image de l'instruction ldl :

Listing 52 –

```
1 ldl n
2 newAREG      word[WPTR @ n]
3 newBREG      AREG
4 newCREG      BREG
```

une affectation d'une variable locale peut se traduire en ST-20 en :

Listing 53 –

```
1 ldc 0
2 stl 0
3
4 ==> local1 = 0
```

L'étape suivante consiste à analyser chaque transputer et le transcrire en code C.

Par exemple le fonctionnement du trasnputer2 peut se traduire en langage C :

Listing 54 –

```
1
2 uint8_t
3  transputer2 (uint8_t * data, int size)
4  {
```

```
5     uint8_t read[4];
6
7   //printf("[+] Calling transputer2 \n");
8
9   // Lecture 12 octet sur le channel 0 <-- x
10    // 12 Octets --> channel 1
11    read[1] = transputer7 (data, 0xC);
12    // 12 octets --> channel 2
13    read[2] = transputer8 (data, 0xC);
14    // 12 octets --> channel 3
15    read[3] = transputer9 (data, 0xC);
16
17    // 1 octet <-- channel 1
18    // 1 octet <-- channel 2
19    // 1 octet <-- channel 3
20
21    read[0] = read[1] ^ read[2] ^ read[3];
22
23
24    return (read[0]);
25
26
27  }
```

L'ensemble des transputers réalise des opérations sur la clé à l'image du transputer2, toutefois le transputer0 se charge aussi de déchiffrer la donnée en entrée :

Ci-dessous un extrait du code C du transputer0 :

Listing 55 –

```
1
2   for (i = 0; i < data_size; i++)
3       {
4   //printf("[+] Pushing Byte[%d] of data \n",i);
5
6
7         //printf(ANSI_COLOR_RED "State %d \n" ANSI_COLOR_RESET, i);
8
9         // lecture 1 octet  sur channel 0 <-- x
10        x = data[i];
11        // 12 Octets --> channel 1
12        read[1] = transputer1 (key, 0xC);
13        // 12 octets --> channel 2
14        read[2] = transputer2 (key, 0xC);
15        // 12 octets --> channel 3
16        read[3] = transputer3 (key, 0xC);
17        //printf("[+] Round %d : sending reading to/from transputers
              ok \n",i);
18
19        read[1] = read[1] ^ read[2] ^ read[3];
20        //printf(" [+]  xoring inputs OK ...\n");
21
22
23        read[0] = x ^ ((2 * key[count]) + count);
24        //printf(" [+]  xoring data OK ...\n");
25
26
27        key[count] = read[1];
28        //printf(" [+]  updating key array OK ...\n");
```

En effet, la variable read[0] va contenir à chaque fois l'octet déchiffré. A la fin de cette étape on s'assure que notre code fonctionne correctement en lui soumettant le vecteur de test fourni avec le fichier "schematics.pdf".

5.1 À la recherche de la clé !

Notre objectif maintenant est d'identifier la clé à utiliser pour déchiffrer le fichier "encrypted". En analysant la fin du code ST-20, on s'aperçoit des élements suivants :

Listing 56 –

```
1
2  ROM:7FF50985 aKey:           db "KEY:"
3  ROM:7FF50989                 db 0FFh
4  ROM:7FF5098A                 db 0FFh
5  ROM:7FF5098B                 db 0FFh
6  ROM:7FF5098C                 db 0FFh
7  ROM:7FF5098D                 db 0FFh
8  ROM:7FF5098E                 db 0FFh
9  ROM:7FF5098F                 db 0FFh
10 ROM:7FF50990                 db 0FFh
11 ROM:7FF50991                 db 0FFh
12 ROM:7FF50992                 db 0FFh
13 ROM:7FF50993                 db 0FFh
14 ROM:7FF50994                 db 0FFh
15 ROM:7FF50995                 db 23
16 ROM:7FF50996                 db 'c', 'o', 'n', 'g', 'r', 'a', 't',
       'u', 'l', 'a', 't'
17 ROM:7FF50996                 db 'i', 'o', 'n', 's', '.', 't', 'a',
       'r', '.', 'b', 'z'
```

Il est à noter que la partie chiffrée commence juste après cet extrait.

La présence d'une chaîne de caractères "KEY :" suivie des valeurs 0xFF (12 octets) nous laisse supposer que c'est à notre charge de retrouver la clé.

Ensuite, on remarque la chaîne de caractères "congratulations.tar.bz2", on peut supposer alors que le fichier "encrypted" une fois déchiffré aura le format ".tar.bz2".

Ça tombe bien car en analysant le comportement du transputer0 on remarque que l'opération suivante , basée sur un xor, impacte les 12 premiers de la clé avant sa modification par les autres transputers :

Listing 57 –

```
1 ...
2 read[0] = x ^ ((2 * key[count]) + count);
3 ...
```

Partant de ce constat, on peut déjà analyser un fichier de type "tar.bz2" et réupérer ses 12 premiers octets :

Listing 58 –

```
1 42 5A 68 39  31 41 59 26  53 59 22 57
```

Néanmoins en lisant la spécification de l'en-tête bz2 [4]on se rend compte que les deux dernier octets correspondent au crc du fichier décompressé. Cela veut dire qu'on peut récupérer que les dix premiers octets de la clé.

Listing 59 –

```
1 .magic:16                     = 'BZ' signature/magic number
2 .version:8                    = 'h' for Bzip2 ('H'uffman coding),
      '0' for Bzip1 (deprecated)
3 .hundred_k_blocksize:8        = '1'..'9' block-size 100 kB-900 kB
      (uncompressed)
```

4. http ://en.wikipedia.org/wiki/Bzip2

```
4
5  .compressed_magic:48            = 0x314159265359 (BCD (pi))
6  .crc:32                         = checksum for this block
7  .randomised:1                   = 0=>normal, 1=>randomised (
       deprecated)
8  .origPtr:24                     = starting pointer into BWT for
       after untransform
9  ...
```

Jusqu'ici tout va bien on écrit une fonction qui ne retourne la clé initiale à partir du chiffré (10 premiers octets d'un fichier "bz2") :

Listing 60 –

```
1
2   KeyFromCipher (uint8_t * key, uint8_t * data, uint8_t * result,
        uint8_t len)
3   {
4     uint8_t i;
5     int test[12];
6     for (i = 0; i < len; i++)
7         result[i] = (((data[i] ^ key[i]) - i) / 2);
8     return 0;
9
10  }
```

On se lance dans un " brute force" des deux derniers octets, sauf qu'on se rend compte que cela n'est pas trop optimisé.

Pour gagner un peu de temps, je décide de ne pas déchiffrer la totalité de la data mais juste les 32 premiers octets et regarder s'il contiennent une suite de 0xFF.

Car si on regarde les 32 premiers octets d'un fichier "bz2" on se rend compte que ces derniers correspondent à une série de 0xFF :

Listing 61 –

```
1
2  00000000   42 5A 68 39  31 41 59 26  53 59 22 57  F7 1F 00 C1  5A 7
       F FF FF  BZh91AY&SY W....Z...
3  00000014   FF FF FF FF  FF FF FF FF  FF FF FF FF  FF FF FF FF  FF
       FF FF FF  ....................
4  00000028   FF FF FF FF  FF FF FF FF  FF FF FF E2  72 F3 B5 BD  CD
       F5 BA EF  ............r.......
5  0000003C   BD 6F BD EA  DF 7B DE F6  B6 FB BB CF  3C 9E EE DE  DF 7
       D BE EE  .o...{......<....}..
6  00000050   57 88 ED DC  FB EE 75 2E  DE 5F 3A D5  D3 D6 F5 A5  F4
       65 74 F7  W.....u._:......et.
```

À ce stade, j'arrive à réduire le temps d'execution de mon programme, sauf que aucun des fichiers sortis ne présente l'empreinte sha256 souhaitée.

Deux hypothèses sont à considérer alors :

la clé n'est pas correcte, ou son calcul,

le code de déchiffrement n'est pas correcte.

Regardons de plus près la clé utilisée et sa formule de calcul, ci dessus la sortie de la fonction qui permet de ressortir la clé à partir de l'en-tête "bz2" et notre chiffré :

Listing 62 –

```
1
2  I love ST20 architecture
3  Clear is
4   fe   f3   50   dc   81   bc   97   27   89   ac   40   0
5  Cipher is
6   42   5a   68   39   31   41   59   26   53   59   1   5e
7  Key is
8   5e   54   1b   71   56   7c   64   fd   69   76   1b   29
```

La première remarque est la présence d'octets présentant le bit poids fort activé (0xfd), et la deuxième remarque concerne la possibilité le fait d'un octet de la clé peut prendre la valeur affiché ci-dessus ou sa valeur + 0x80 pour arrivé au chiffré.

On se retrouve face à une deuxième diificulé, de plus le brute force il faut tester la valeur de chaque octet de la clé et la valeur + 0x80 .

La fonction main de notre devient comme suit :

Listing 63 –

```
1  for (i=0 ; i< 10 ; i++)
       key_orig[i] = key_orig[i] & 0x7f;
2
3    uint8_t key[12];
4    uint8_t key_tmp[12];
5    int counter1, counter2;
6
7    CheckCode (data_vector, key_vector, 24, 12, recipher);
8
9
10 //read data from data.enc
11
12   FILE *fl = fopen ("data.enc", "rb");
13   fseek (fl, 0, SEEK_END);
14   long len = ftell (fl);
15   uint8_t *ret = malloc (len);
16   fseek (fl, 0, SEEK_SET);
17   fread (ret, 1, len, fl);
18   fclose (fl);
19
20   //printf ("\n[+] Calling main() \n");
21   for (counter1 = 0; counter1 < 1024; counter1++)
22     {
23       for (counter2 = 0; counter2 < 10; counter2++)
24         {
25           key_tmp[counter2] =
26             key_orig[counter2] + ((counter1 & (0x1 << counter2)) ?
                   0x80 : 0);
27
28         }
29       printf(" key combination %d  " , counter1 );
30       printTable (key_tmp, 12);
31       for (x = 0x00; x <= 0xFF; x++)
32         {
33           for (y = 0x00; y <= 0xFF; y++)
34             {
35               key_tmp[10] = (x & 0xFF);
36               key_tmp[11] = (y & 0xFF);
37               memcpy (key, key_tmp, 12);
38
39               if ((checksignature (key, ret)) == 1)
40                 {
```

```
41                      CheckCode (data_vector, key_vector, 24, 12,
                            recipher);
42
43                      init ();
44  ...
```

Je me retrouve avec les trous fichiers suivants :

Listing 64 –

```
1
2  sha256sum *.bin
3  daeacde0803bd891edc883aee5da6ebd6f37fa4fea5a8d1d89da3eb6250c6928
       dec7a-6e-29cf1625.bin
4  ec538c971d502221e989d38083d729a1039472dfe64eae748ac1de254653d341
       dec7a-6e-5f64f3ca.bin
5  9128135129d2be652809f5a1d337211affad91ed5827474bf9bd7e285ecef321
       decda-c5-66170ede.bin
```

La troisième ligne ci-dessus correspond bien à ce qu'on recherche et on termine ainsi le stage 5.

6 La ligne d'arrivée !

On commmence notre sixième niveau par le fichier "congratulations.tar.bz2", qui une fois décompressé nous permet de récupérer le fichier suivant :

Listing 65 –

```
1
2  > file congratulations.jpg
3  congratulations.jpg: JPEG image data, JFIF standard 1.01
4
5  > sha256sum congratulations.jpg
6  8b381121312a5941cdee21447bc525f3add35af0792e5d029ed3bb6b51d00ca5
       congratulations.jpg
```

Un coup de binwalk :

Listing 66 –

```
> binwalk congratulations.jpg
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------

0             0x0             JPEG image data, JFIF standard 1.01
55248         0xD7D0          bzip2 compressed data, block size =
    900k
```

L'extraction du fichier "bzip2" embarqué se fait avec l'option "-e" de binwalk ce qui nous permet de retrouver une deuxième image :

Listing 67 –

```
>file congratulations.png
congratulations.png: PNG image data, 636 x 474, 8-bit/color RGBA,
    non-interlaced
```

Un deuxième coup de binwalk :

Listing 68 –

```

> binwalk congratulations.png

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------

0             0x0             PNG image, 636 x 474, 8-bit/color
    RGBA, non-interlaced
133286        0x208A6         Zlib compressed data, best
    compression
```

L'extraction de la partie compressée avec Zlib ne donne pas de résultat probant, pour cela on vérifie la structure de l'image avec pngcheck :

Listing 69 –

```
pngcheck -v -f 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 0x013a2, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x026e5, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x03a28, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x04d6b, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x060ae, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x073f1, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x08734, length 4919:  illegal reserved-bit-
      set chunk
```

```
  chunk sTic at offset 0x09a77, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x0adba, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x0c0fd, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x0d440, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x0e783, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x0fac6, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x10e09, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x1214c, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x1348f, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x147d2, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x15b15, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x16e58, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x1819b, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x194de, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x1a821, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x1bb64, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x1cea7, length 4919:  illegal reserved-bit-
      set chunk
  chunk sTic at offset 0x1e1ea, 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 0x228ae, length 8192
  chunk IDAT at offset 0x248ba, length 8192
  chunk IDAT at offset 0x268c6, length 8192
  chunk IDAT at offset 0x288d2, length 8192
  chunk IDAT at offset 0x2a8de, length 8192
  chunk IDAT at offset 0x2c8ea, length 8192
  chunk IDAT at offset 0x2e8f6, length 6827
  chunk IEND at offset 0x303ad, length 0
ERRORS DETECTED in congratulations.png
```

Ok, on note la présence du chunk "sTic", l'idée est d'extraire ces chunks et les rassembler, je vous épargne le code utilisé (TweakPng ou pypng) pour cette étape. L'en-tête du fichier reconstruit est le suivant :

Listing 70 –

```
00000000   78 9C 84 B6  7B 38 13 EE  FB 38 3E CC  DA 44 D9 0C  DB
    54 B6 D9  D6 26 95 CD
```

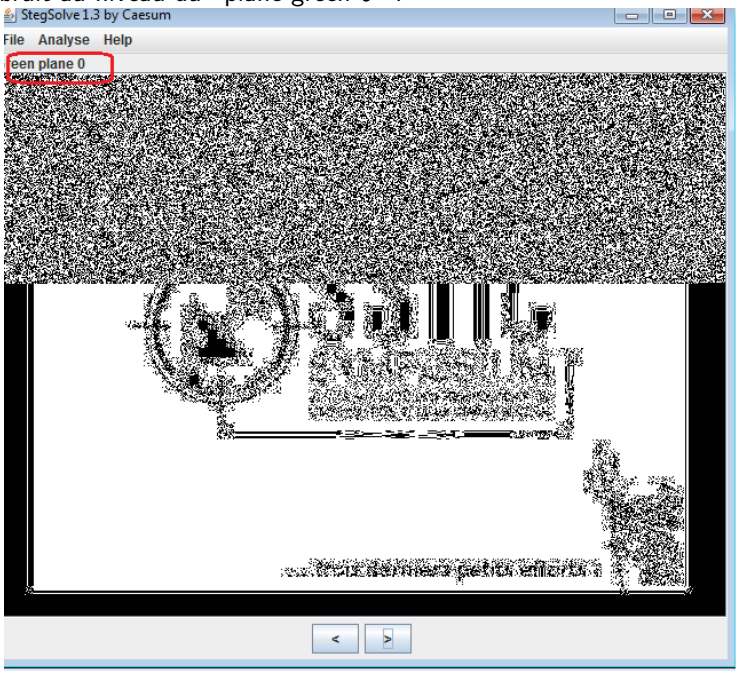
on refait un coup de binwalk en demandant l'extraction de contenu zlib et on arrive au fichier suivant :

Listing 71 –

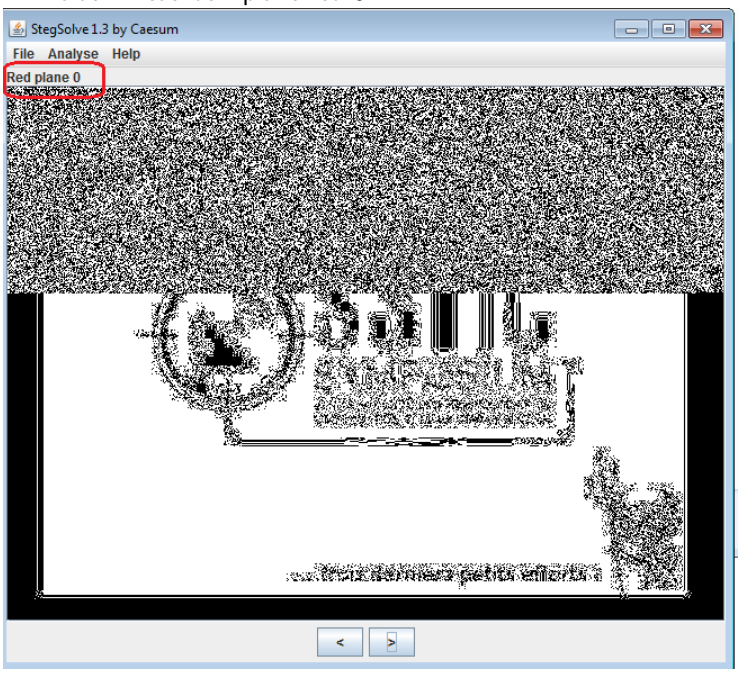
```
1
2 >file _stage6_3.extracted/0
3 _stage6_3.extracted/0: bzip2 compressed data, block size = 900k
```

Le contenu décompressé correspond au fichier congratulations.tiff qu'on renome en congratulations.png et qu'on passe à StegSolve :

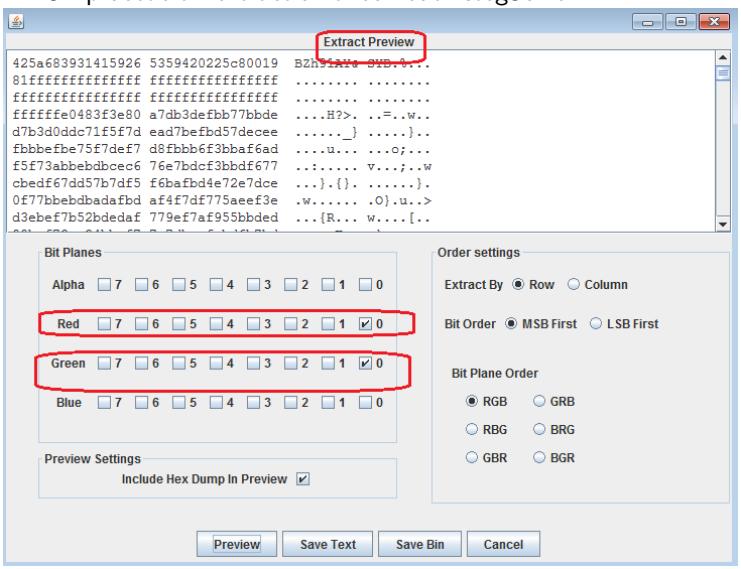
En parcourant les différents "plane" avec l'outil stegSolve, on note la présence de bruit au niveau du "plane green 0" :



Et au niveau du "plane red 0" :



On procède à l'extraction avec l'outil stegSolve :

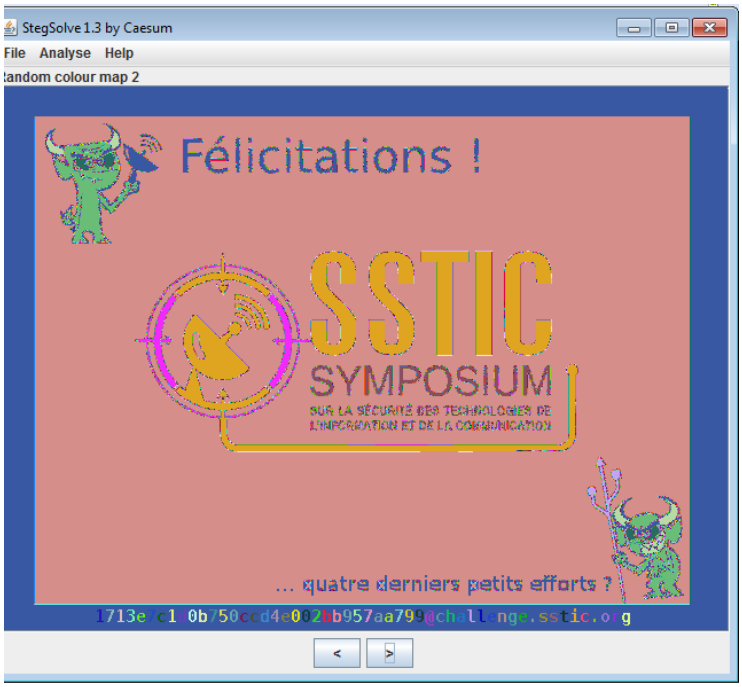


Le fichier obtenu correspond à un fichier tar :

Listing 72 –

```
> file _stage6_3.extracted/red_green0.out
_stage6_3.extracted/red_green0.out: POSIX tar archive (GNU)
```

Son extraction permet de récupérer le fichier "congratulations.gif" qu'on convertit encore une fois en png puis on le passe à stegSolve. En parcourant les différents "plane" on tombe sur l'adresse mail recherchée :


StegSolve 1.3 by Caesum
File Analyse Help
Félicitations !
SSTIC
SYMPOSIUM
... quatre derniers petits efforts ?
1713e[illegible]c1[illegible]0b750ccd4e002bb957aa799@challenge.sstic.org
<
>