Challenge SSTIC 2015 (solution)

Romain CARRÉ <romain.carre@cea.fr>

16 mai 2015

Table des matières

1 Introduction **1**
1.1 Énoncé . . . 1
1.2 Préambule . . . 1
1.3 Remerciements . . . 2

2 Résolution **2**
2.1 Stage 1 - DuckyScript . . . 2
2.2 Stage 2 - OpenArena . . . 5
2.2.1 Jouer au jeu réellement . . . 8
2.2.2 Jouer au jeu en mode développeur . . . 8
2.2.3 Jouer au jeu avec les commandes . . . 9
2.2.4 Analyser le fichier de map . . . 12
2.2.5 Bruteforcer la clef . . . 14
2.3 Stage 3 - USB Paint . . . 15
2.4 Stage 4 - Désobscurcissement JS . . . 20
2.5 Stage 5 - Reverse ST20 . . . 27
2.5.1 Avec IDA Free . . . 28
2.5.2 Avec Miasm2 . . . 46
2.5.3 Commentaire sur jserver . . . 58
2.5.4 Commentaire sur st20emu . . . 59
2.6 Stage 6 - Stéganographie . . . 59

A Miasm2 Pull Request **67**

1 Introduction

1.1 Énoncé

> Le défi consiste à analyser la carte microSD qui était insérée dans une clé USB étrange. L'objectif est d'y retrouver une adresse e-mail (@challenge.sstic.org). L'image disque de cette carte microSD est disponible ici : `http://static.sstic.org/challenge2015/challenge.zip`.
>
> `SHA256: bd0df75a1d6591e01212e6e28848aec94b514e17e4ac26155696a9a76ab2ea31 - challenge.zip`
>
> L'équipe d'organisation tient à rappeler qu'aucun matériel spécifique n'est nécessaire à la résolution du challenge.

1.2 Préambule

Ce document a pour but de présenter une démarche possible pour résoudre le challenge SSTIC 2015. Nous aborderons en détails les situations où plusieurs chemins de résolution sont envisageables, et nous expliquerons le processus de création de certaines heuristiques qui permettent parfois de gagner du temps.

Tout le code utilisé dans le cadre de la résolution de ce challenge a été déposé sur le dépôt GitHub `https://github.com/rom1sqr/SSTIC-2015`.

1.3 Remerciements

Je tiens à remercier chaleureusement les concepteurs de ce challenge, qui m'ont permis de passer plusieurs soirées fort intéressantes. Je remercie également le comité d'organisation et le comité de programme du SSTIC. J'adresse enfin un remerciement tout particulier à l'attention de l'équipe sécurité du CEA, que je salue au passage.

2 Résolution

2.1 Stage 1 - DuckyScript

Nous commençons par récupérer l'image disque, vérifier son empreinte et sa validité :

```
$ wget -q http://static.sstic.org/challenge2015/challenge.zip
$ S1_SHA256=bd0df75a1d6591e01212e6e28848aec94b514e17e4ac26155696a9a76ab2ea31
$ echo "$S1_SHA256 *challenge.zip" > SHA256SUMS
$ sha256sum -c SHA256SUMS
challenge.zip: OK
$ unzip -t challenge.zip
Archive:  challenge.zip
    testing: sdcard.img               OK
No errors detected in compressed data of challenge.zip.
```

Il est temps de l'extraire, de la monter localement et de récupérer son contenu :

```
$ unzip challenge.zip
Archive:  challenge.zip
  inflating: sdcard.img
$ sudo mount sdcard.img /mnt
$ ls -al /mnt
total 33452
-rwxr-xr-x 1 root root 34253730 mars  26 02:49 inject.bin
$ cp /mnt/inject.bin .
$ sudo umount /mnt
```

Le contenu n'est pas très explicite en lui-même, mais on suppose qu'il s'agit d'instructions (extension .bin). De prime abord, les motifs répétés en colonnes m'ont fait penser que les opcodes étaient plus ou moins alignés (comme sur plusieurs architectures bien connues), mais cette supposition s'est avérée fausse par la suite.

```
$ hexdump -C inject.bin | head -10
00000000  00 ff 00 ff 00 ff 00 ff  00 ff 00 ff 00 ff 00 d7  |................|
00000010  15 08 00 ff 00 f5 28 00  00 ff 00 ff 00 ff 00 eb  |......(.........|
00000020  06 00 10 00 07 00 28 00  00 32 13 00 12 00 1a 00  |......(..2......|
00000030  08 00 15 00 16 00 0b 00  08 00 0f 00 0f 00 2c 00  |..............,.|
00000040  2d 00 08 00 11 00 06 00  2c 00 1d 02 0a 00 05 02  |-.......,.......|
00000050  1e 00 04 02 0a 02 21 00  04 02 1c 02 1a 00 05 02  |......!.........|
00000060  27 00 04 02 0a 02 0e 00  04 02 05 00 1a 00 05 02  |'...............|
00000070  18 00 04 02 06 02 04 02  04 02 07 00 1a 00 05 02  |................|
00000080  1c 00 04 02 0a 02 0e 00  04 02 07 00 04 02 05 02  |................|
00000090  0f 00 04 02 09 02 25 00  04 02 1d 02 0a 00 05 02  |......%.........|
```

L'image disque est assez volumineuse (123 Mo), mais une rapide analyse des chaînes de caractères présentes nous donne un indice intéressant :

```
$ strings -a sdcard.img | tail -1
java -jar encoder.jar -i /tmp/duckyscript.txt
```

Après une recherche sur le net, on constate qu'il existe un langage de script (le DuckyScript [1]) qui permet d'interagir avec l'environnement Windows, qui peut être compilé, et qui est probablement à l'origine du binaire que nous avons sous les yeux. Il faut alors vérifier si on peut le décompiler. En analysant le code source de l'encoder Java [2], il semble que le processus de compilation est aisément réversible. Au moment de la résolution de cette partie du challenge, je ne savais pas encore qu'il existait

1. DuckyScript : `https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Duckyscript`
2. `https://github.com/hak5darren/USB-Rubber-Ducky/blob/master/Encoder/src/Encoder.java`

un outil de décompilation du DuckyScript (`ducky-decode.pl`[3]), j'en ai donc implémenté un (du moins sur le sous-ensemble des instructions qui nous intéresse ici).

Listing 1 – src/stage1/duckdecode.py

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

# https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Duckyscript

"DuckyScript decompiler"

__version__ = '0.1'

import sys

_BYTE_TO_CHAR_MAP = {
    0: '\x00'*0x27 + '0\x00\x00\x00\x00 -=[]\\\x00;\'`,./',
    2: '\x00'*0x1E + '!@#$%^&*()\x00\x00\x00\x00\x00_+{}|\x00:"~<>?',
}

def byte_to_char(byte, shift = 0):
    "charToByte (Encoder.java:416) inverse function"
    if 4 <= byte <= 29:
        return chr(byte + 0x5D - shift * 0x10)
    if 30 <= byte <= 38 and shift == 0:
        return chr(byte + 0x13)
    return _BYTE_TO_CHAR_MAP[shift][byte]

def decode_to_stdout(filename):
    "encodeToFile (Encoder.java:121) inverse function"
    data = open(filename, 'rb').read()
    i, maxi = 0, len(data)
    delay = 0
    string = ''
    while i < maxi:
        if data[i] == '\x00':
            delay += ord(data[i + 1])
        elif delay:
            if string:
                print 'STRING %s' % string
                string = ''
            print 'DELAY %d' % delay
            delay = 0
            continue
        elif data[i + 1] == '\x08':
            if string:
                print 'STRING %s' % string
                string = ''
            print 'WINDOWS %s' % byte_to_char(ord(data[i]))
        elif data[i] == '\x28' and data[i + 1] == '\x00':
            if string:
                print 'STRING %s' % string
                string = ''
            print 'ENTER'
        elif data[i + 1] in '\x00\x02':
            string += byte_to_char(ord(data[i]), ord(data[i + 1]))
        else:
            raise ValueError(data[i : i + 10].encode('hex'))
        i += 2

if __name__ == '__main__':
    decode_to_stdout(sys.argv[1])
```

Nous l'utilisons donc directement :

```
$ python duckdecode.py inject.bin > inject.duck
```

Voyons ce que ce script contient, et les actions qu'il effectue :

3. `https://code.google.com/p/ducky-decode/source/browse/trunk/ducky-decode.pl`

```
$ head -9 inject.duck
DELAY 2000
WINDOWS r
DELAY 500
ENTER
DELAY 1000
STRING cmd
ENTER
DELAY 50
STRING powershell -enc ZgB1AG4AYwB0AGkAbwBuA[...]ApADsAfQA=
```

Le raccourci clavier `WINDOWS+R` sous Windows affiche la fenêtre issue du menu Démarrer/Exécuter, permettant d'exécuter un binaire rapidement à partir de l'interface graphique. Le binaire `cmd` est l'interpréteur de commandes Windows. Et cette série de touche est suivie de plusieurs commandes PowerShell [4] (un shell évolué pour Windows). La série de caractères alphanumériques et le signe "=" en fin de chaîne nous font immédiatement penser que les commandes qui sont adressées au shell sont encodées en Base64. En effet, d'après la documentation :

```
$ powershell /?

-EncodedCommand
    Accepte la version encodée en Base64 d'une commande. Utilisez ce
    paramètre pour envoyer à Windows PowerShell des commandes qui
    nécessitent des guillemets anglais complexes ou des accolades.
```

Il faut donc les décoder pour voir ce qu'elles contiennent. De la même manière, nous allons implémenter un petit script pour automatiser cette opération. On notera au passage que les caractères finalement obtenus sont encodés en UTF-16.

Listing 2 – src/stage1/psdecode.py

```
1  #!/usr/bin/env python
2  #-*- coding:utf-8 -*-
3
4  "Base64-decode powershell commands in DuckyScript source file"
5
6  __version__ = '0.1'
7
8  import sys
9
10 with open(sys.argv[1], 'rb') as duckfile:
11     for line in duckfile:
12         if line.startswith('STRING'):
13             cmd = line.split(' ', 1)[1]
14             if cmd.startswith('powershell'):
15                 print cmd.split()[2].decode('base64').decode('utf-16')
```

Et nous l'utilisons :

```
$ python psdecode.py inject.duck > inject.ps
$ sed 's/{.*//g' inject.ps | sort -u
function hash_file
function write_file_bytes
$ head -1 inject.ps | sed 's/;/;\n/g;s/){/){\n/g' | grep write
function write_file_bytes{param([Byte[]] $file_bytes, [string] $file_path = ".\stage2.zip");
write_file_bytes([Convert]::FromBase64String('UEsDBAoDAAAAADaK[...]W2UwdXtOh6gUsBzWnXw=='));
$ tail -1 inject.ps | sed 's/;/;\n/g'
function hash_file{param([string] $filepath);
$sha1 = New-Object -TypeName System.Security.Cryptography.SHA1CryptoServiceProvider;
$h = [BitConverter]::ToString($sha1.ComputeHash([System.IO.File]::ReadAllBytes($filepath)));
$h}$h = hash_file(".\stage2.zip");
if($h -eq "EA-9B-8A-6F-5B-52-7E-72-65-20-19-31-3C-25-B5-6A-D2-7C-7E-C6"){echo "You WIN";
}else{echo "You LOSE";}
```

À l'intérieur de ce script PowerShell, on note l'usage de deux types de fonctions : `write_file_bytes` et `hash_file`, qui intuitivement écrivent de la donnée dans un fichier et calculent son empreinte (SHA1

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

dans notre cas). Il s'agit donc maintenant d'extraire la donnée brute, de la stocker dans un fichier et de vérifier que le hash calculé correspond bien à celui attendu.

Nous implémentons alors un nouveau script qui est chargé d'automatiser ce procédé :

Listing 3 – src/stage1/psexec.py

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

"Execute PowerShell commands in a script file"

__version__ = '0.1'

import sys
import re
import hashlib

REGEXP_WFB = re.compile(
    r"function write_file_bytes.*::FromBase64String\('([^']+)'.*else")
REGEXP_HF  = re.compile(
    r'function hash_file.*-eq "([^"]+)"')

def main():
    "main function"
    data, h4sh, outfile = '', '', 'stage2.zip'

    with open(sys.argv[1], 'rb') as psfile:
        for line in psfile:
            match_wfb = REGEXP_WFB.match(line)
            if match_wfb:
                data += match_wfb.group(1).decode('base64')
                continue
            match_hf  = REGEXP_HF.match(line)
            if match_hf:
                h4sh  = match_hf.group(1).replace('-','').lower()
                continue
            raise ValueError('cannot interpret this line')

    if hashlib.sha1(data).hexdigest() == h4sh:
        open(outfile, 'wb').write(data)
        print >> sys.stderr, "OK", outfile
    else:
        print >> sys.stderr, "FAIL"

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

Et nous l'utilisons :

```
$ python psexec.py inject.ps
OK stage2.zip
```

Nous obtenons une archive ZIP avec l'étape suivante du challenge.

2.2 Stage 2 - OpenArena

Nous pouvons commencer par copier le stage 2 dans un répertoire dédié, vérifier son intégrité, et l'extraire :

```
$ mkdir -p stage2
$ cp stage2.zip stage2/
$ cd stage2
$ unzip -t stage2.zip
Archive:  stage2.zip
    testing: encrypted                OK
    testing: memo.txt                 OK
    testing: sstic.pk3                OK
No errors detected in compressed data of stage2.zip.
$ unzip stage2.zip
```

```
Archive :  stage2.zip
 extracting : encrypted
  inflating : memo.txt
  inflating : sstic.pk3
$ rm -f stage2.zip
$ file *
encrypted : data
memo.txt :  ASCII text
sstic.pk3 : Zip archive data, at least v2.0 to extract
```

Nous observons trois fichiers : de la donnée brute, un fichier texte et une autre archive ZIP. Commençons par lire le fichier texte :

```
$ cat memo.txt
Cipher : AES-OFB
IV : 0x53535449433230313532d537461676532
Key : Damn... I ALWAYS forget it. Fortunately I found a way to hide it into my favorite game !

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

On comprend ici que le fichier encrypted est un fichier chiffré en AES, avec le mode d'opération OFB (*Output FeedBack*[5]). On dispose également du hash SHA256 du fichier chiffré, et du fichier déchiffré.

Vérifions tout d'abord que le hash du fichier chiffré est correct :

```
$ S2_SHA256=91d0a6f55cce427132fc638b6beecf105c2cb0c817a4b7846ddb04e3132ea945
$ echo "$S2_SHA256 *encrypted" > SHA256SUMS
$ sha256sum -c SHA256SUMS
encrypted : OK
```

L'IV (Initialization Vector[6]) est fourni, il ne manque que la clef, que l'on doit apparemment retrouver dans le jeu favori du concepteur. Notre intuition nous amène à extraire l'archive `sstic.pk3` que nous avons à notre disposition :

```
$ unzip -d game sstic.pk3
```

Et globalement, voici à quoi ressemble l'arborescence :

```
$ tree game
```

5. http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Output_Feedback_.28OFB.29
6. http://en.wikipedia.org/wiki/Initialization_vector

```
game
├── AUTHORS
├── levelshots
│   └── sstic.tga
├── maps
│   └── sstic.bsp
├── README
├── scripts
│   └── sstic.arena
├── sound
│   └── world
│       └── bj3.wav
└── textures
    └── sstic
        ├── 01.tga
        ├── 02.tga
        ├── 103336131.tga
        ├── 1036082074.tga
        ├── 1039223422.tga
        ├── 1042015984.tga
        ├── 1065969731.tga
        ├── 1068346222.tga
        ├── 110183801.tga
        ├── ⋮
        ├── 883915618.tga
        ├── 902610813.tga
        ├── 905737444.tga
        ├── 90736223.tga
        ├── 928614786.tga
        ├── 968350285.tga
        ├── 976571476.tga
        ├── 994048089.tga
        └── logo.tga
```

De quoi s'agit-il ? Commençons par regarder ce que contiennent les fichiers README et AUTHORS :

```
$ cat game/README
Copy the pk3 in your baseoa directory.
In the game, open the console (²) and type \map sstic.
$ cat game/AUTHORS
Icons
Open Iconic v1.1.1 — useiconic.com

Maps
085am_underworks2 v0.8.5 by Neon_Knight — openarena.wikia.com
```

On dispose d'un lien vers `openarena.wikia.com`, qui après consultation sous-entend que nous sommes en présence d'une implémentation libre et multiplateforme d'un moteur de jeu multijoueur basé sur le célèbre Quake 3 Arena, à savoir : *OpenArena*.

Vérifions à quoi servent les autres fichiers.

Tous les fichiers TGA sont des images (logo ou textures) et le fichier WAV est un fichier audio.

```
$ file game/maps/sstic.bsp
game/maps/sstic.bsp : Quake III Map file (BSP)
```

Le fichier BSP contient les informations sur la carte "sstic".

```
$ cat game/scripts/sstic.arena
{
map                     "sstic"
longname                "SSTIC Challenge"
fraglimit               "25"
bots                    "ayumi sarge kyonshi beret s_marine"
type                    "ffa lms team elimination dom"
}
```

Le fichier ARENA contient des constantes concernant la carte "sstic".

Si on observe rapidement les fichiers de textures, on remarque plusieurs portions de hash, nombreuses, toujours en trois couleurs : orange, blanc, ou vert. On devine qu'elles pourraient constituer les octets de la clef. Impossible dans l'immédiat de savoir quelles portions doivent être conservées, ni dans quel ordre il faut les placer. En voici trois, à titre d'exemple :

À partir de cet instant, il existe plusieurs pistes d'exploration possibles :

1. jouer au jeu réellement et espérer trouver la clef ;
2. jouer au jeu en trichant et espérer trouver la clef ;
3. utiliser certaines commandes du jeu et obtenir des informations ;
4. analyser le fichier BSP pour récupérer la clef ;
5. bruteforcer la clef avec toutes les portions de hash.

2.2.1 Jouer au jeu réellement

C'est la première idée qui m'a traversé l'esprit. On ne sait pas trop quoi chercher, ni où, ça semble être la moins mauvaise solution. Il faut donc installer le serveur et le client de jeu, et copier la carte. Sous Debian :

```
$ apt-get install openarena openarena-server
$ cp stage2/sstic.pk3 ~/.openarena/baseoa/
$ /usr/games/openarena &
```

Après avoir passé les différents menus, nous arrivons sur la carte, où l'on remarque à certains endroits que les textures des murs, ou des objets environnants ont été remplacées par les textures custom comme celles d'au-dessus. On comprend donc immédiatement qu'il va falloir plus ou moins rechercher toutes les textures custom qui sont affichées dans la carte.

Après plusieurs tentatives sur les interrupteurs, les timers qui ne nous facilitent pas la tâche, les multiples pièges mortels (comme la mort instantanée ou les piscines de lave), et les manoeuvres compliquées de Rocket Jump [7], on est finalement téléporté dans une pièce qui arbore ce fronton :

On se remémore alors que les tuiles custom comportent un petit symbole, et que la couleur du tronçon de hash est indiquée par la couleur de l'indicateur. On suppose que l'ordre des panneaux indicateurs donne l'ordre des tronçons. Le principal problème à cet instant, c'est que je n'ai pas réussi à trouver toutes les tuiles custom dans la carte. Il va donc probablement falloir utiliser un autre moyen...

2.2.2 Jouer au jeu en mode développeur

Comme je n'arrivais pas à localiser toutes les tuiles custom, et voulant par la même m'économiser la frustration d'autres rocket jumps infructueux, j'ai décidé d'abandonner mon estime de moi-même en utilisant le mode développeur du moteur openarena. La manipulation est assez simple :

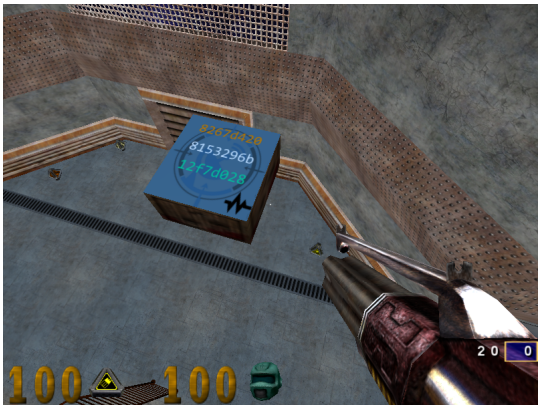
```
$ /usr/games/openarena +set sv_cheats 1 &
```

puis dans la console du jeu (touche ~) :

7. http://en.wikipedia.org/wiki/Rocket_jumping

```
/devmap sstic
/god
/noclip
```

Nous ne sommes alors plus soumis à la gravité, nous sommes invincibles et nous pouvons traverser les murs (ou tout autre objet sujet à collision). Mais ce fut la déception : impossible de découvrir davantage de tuiles custom.



Il a donc fallu songer à une autre solution.

2.2.3 Jouer au jeu avec les commandes

On demande alors la liste des commandes disponibles[8] dans la console du jeu. Deux d'entre elles semblent très intéressantes :

```
/imagelist
/condump imagelist.log
```

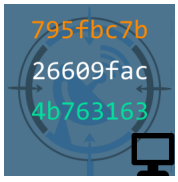
8. http://openarena.wikia.com/wiki/Command_console

La première liste toutes les textures chargées par le jeu (ou que le jeu a essayé de charger) pour être affichées dans la carte courante ; la seconde exporte tout le contenu de la console courante dans un fichier sur le disque.

Le résultat est assez rapidement obtenu, et nous pouvons lister les images qui sont chargées :

- 01.tga
- 02.tga
- 19281330.tga
- 31570422.tga (un soleil)
- 52809216.tga (un lien)
- 71921642.tga (une goutte)
- 86520831.tga (symbole wifi)
- 123771438.tga
- 381650771.tga
- 457615433.tga (un drapeau)
- 643008245.tga
- 747908186.tga
- 838783866.tga (un écran)
- 855646320.tga (une disquette)
- 1219191984.tga
- 1429015180.tga
- 1509983054.tga (une goutte)
- 1604401524.tga (une disquette)
- 1749623519.tga
- 1773100136.tga (une onde)
- 2036414783.tga
- logo.tga

Problème : il y en a un peu trop : 2 disquettes et 2 gouttes. Mais l'espace de clef est néanmoins suffisamment réduit pour effectuer un bruteforce !

 457615433.png	Pour le drapeau vert, il n'y a qu'un seul drapeau de chargé. Le tronçon de hash est donc `9e2f31f7`.
 1773100136.png	Pour l'onde blanche, il n'y a qu'une seule onde de chargée. Le tronçon de hash est donc `8153296b`.
 855646320.png  1604401524.png	Pour la disquette orange, il y a deux disquettes de chargées. Le tronçon de hash est donc soit `34a19826`, soit `3d9b0ba6`.
 71921642.png  1509983054.png	Pour la goutte blanche, il y a deux gouttes de chargées. Le tronçon de hash est donc soit `7695dc7c`, soit `a5cb854f`.
 457615433.png	Pour le drapeau orange, il n'y a qu'un seul drapeau de chargé. Le tronçon de hash est donc `b0daf152`.
 52809216.png	Pour le lien vert, il n'y a qu'un seul lien de chargé. Le tronçon de hash est donc `b54cdc34`.
 86520831.png	Pour le wifi vert, il n'y a qu'un seul wifi de chargé. Le tronçon de hash est donc `ffe0d355`.
 838783866.png	Pour l'écran blanc, il n'y a qu'un seul écran de chargé. Le tronçon de hash est donc `26609fac`.

On peut donc implémenter un script pour automatiser la tâche :

Listing 4 – src/stage2/decode.py

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

"Stage2 AES-OFB decoder"

__version__ = '0.1'

import Crypto.Cipher.AES
import hashlib

IV        = '53535449433230313352d537461676532'.decode('hex')
SHA256_E = '91d0a6f55cce427132fc638b6beecf105c2cb0c817a4b7846ddb04e3132ea945'
SHA256_D = '845f8b000f70597cf55720350454f6f3af3420d8d038bb14ce74d6f4ac5b9187'

def main():
    "main function"
    with open('encrypted', 'rb') as encrypted:
        data = encrypted.read()
        assert(hashlib.sha256(data).hexdigest() == SHA256_E)
        for key2 in ['34a19826', '3d9b0ba6']:
            for key3 in ['7695dc7c', 'a5cb854f']:
                key = ('9e2f31f7' + '8153296b' + key2 + key3 +
                       'b0daf152' + 'b54cdc34' + 'ffe0d355' + '26609fac')
                cipher = Crypto.Cipher.AES.new(
                    key.decode('hex'),
                    Crypto.Cipher.AES.MODE_OFB, IV)
                msg = cipher.decrypt(data)
                msg = msg.rstrip('\x10')    # /!\ REMOVE PADDING BEFORE HASH !
                if hashlib.sha256(msg).hexdigest() == SHA256_D:
                    print 'OK'
                    open('decrypted', 'wb').write(msg)

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

⚠ **Attention !** Les concepteurs ont choisi de calculer le hash **après** retrait du *padding* (ce qui peut valoir de longues minutes de debugging).

Comme précédemment, le fichier déchiffré est une archive ZIP qui constituera notre stage 3.

```
$ file decrypted
decrypted: Zip archive data, at least v1.0 to extract
$ cp decrypted ../stage3.zip
$ cd ..
```

2.2.4 Analyser le fichier de map

À ce stade ce n'est plus nécessaire, mais il était également possible de procéder ainsi. La spécification du format BSP est relativement bien décrite sur le net [9].

9. http://www.mralligator.com/q3/

Layout of an IBSP file :

Header / Directory
Lump
Lump
Lump
...

header
string[4] *magic*
int *version*
direntry[17] *direntries*
direntry
int *offset*
int *length*

Index	Lump Name
0	Entities
1	Textures
2	Planes
3	Nodes
...	...
11	Meshverts
12	Effects
13	Faces
14	Lightmaps
15	Lightvols
16	Visdata

Il était possible d'accéder aux deux sous-structures intéressantes : le 1-Textures, et 13-Faces. Il suffisait de lister les textures qui sont réellement chargées par le moteur de jeu et de vérifier que chacune d'elle était bien reliée à une face (surface). À titre purement indicatif, voici un PoC écrit en Python :

Listing 5 – src/stage2/bsp-parse.py

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

from struct import unpack_from
from glob   import iglob
from PIL    import Image

d = open('game/maps/sstic.bsp', 'rb').read()

direntry = {}
print 'magic   = %r'      % unpack_from('4s', d[0:])[0]
print 'version = 0x%02x'  % unpack_from('<I', d[4:])[0]
for i in xrange(17):
    offset = unpack_from('<I', d[8*i+ 8:8*i+12])[0]
    length = unpack_from('<I', d[8*i+12:8*i+16])[0]
    direntry[i] = {'offset': offset, 'length': length}
    print 'direntry[%2d]: ' % i,
    print 'offset=%08X' % direntry[i]['offset'],
    print 'length=%d'   % direntry[i]['length']

# récupère les noms des fichiers de textures custom
diskfiles = []
for i in iglob('game/textures/sstic/*.tga'):
    img = Image.open(i)
    if  img.size == (256, 256):
        diskfiles.append(i[5:-4])

# récupère les indices de textures leur correspondant
textures = []
offset, length = direntry[1]['offset'], direntry[1]['length']
data = d[offset:offset+length]
for i in xrange(length / 72):
    texture = data[72*i:72*(i+1)]
    name    = unpack_from('64s', texture[0:])[0]
    name    = name[:name.index('\x00')]
    if  name in diskfiles:
        textures.append(i)
    if  False:
        print 'texture[%d]' % i
        print '\tname     = %r' % name
        print '\tflags    = %d' % unpack_from('<I', texture[64:])[0]
        print '\tcontents = %d' % unpack_from('<I', texture[68:])[0]
print '\033[33mTEXTURES\033[m:', textures

# et filtre celles qui sont dans au moins une face
infaces = set()
offset, length = direntry[13]['offset'], direntry[13]['length']
```

```
48 data = d[offset:offset+length]
49 for i in xrange(length / 104):
50     face    = data[104*i:104*(i+1)]
51     texture = unpack_from('<I', face[0:])[0]
52     if  texture in textures:
53         infaces.add(texture)
54     if  False:
55         print 'face[%d]' % i
56         print '\ttexture = %d' % texture
57 infaces = sorted(list(infaces))
58 print '\033[33mINFACES\033[m: ', infaces
59
60 # on liste les résultats
61 offset, length = direntry[1]['offset'], direntry[1]['length']
62 data = d[offset:offset+length]
63 for i in xrange(length / 72):
64     texture = data[72*i:72*(i+1)]
65     name    = unpack_from('64s', texture[0:])[0]
66     name    = name[:name.index('\x00')]
67     if  i in infaces:
68         realpath = 'game/%s.tga' % name
69         print 'texture[%d]' % i, '\t',
70         print ('name: %r' % name).ljust(35),
71         print  'path: %s' % realpath
```

En le lançant, on obtient exactement la même liste de tuiles custom que par la console en jeu :

```
$ ./bsp-parse.py
magic   = 'IBSP'
version = 0x2e
direntry[ 0]:  offset=00597DAC length=14261
direntry[ 1]:  offset=000000DC length=14256
direntry[ 2]:  offset=0000388C length=544576
direntry[ 3]:  offset=0008F87C length=20592
direntry[ 4]:  offset=000887CC length=28848
direntry[ 5]:  offset=001084F4 length=23748
direntry[ 6]:  offset=0010E1B8 length=28888
direntry[ 7]:  offset=00115290 length=1120
direntry[ 8]:  offset=000948EC length=60456
direntry[ 9]:  offset=000A3514 length=413664
direntry[10]:  offset=001156F0 length=974996
direntry[11]:  offset=0059B564 length=66792
direntry[12]:  offset=0059B564 length=0
direntry[13]:  offset=00203784 length=312936
direntry[14]:  offset=002527AC length=3145728
direntry[15]:  offset=005527AC length=284160
direntry[16]:  offset=0024FDEC length=10688
TEXTURES: [10, 11, 12, 13, 14, 15, 16, 17, [...], 80, 81, 94, 96, 97, 122, 123, 138, 193]
INFACES:  [42, 80, 81, 94, 96, 97, 122, 123, 138, 193]
texture[42]     name: 'textures/sstic/52809216'     path: game/textures/sstic/52809216.tga
texture[80]     name: 'textures/sstic/855646320'    path: game/textures/sstic/855646320.tga
texture[81]     name: 'textures/sstic/457615433'    path: game/textures/sstic/457615433.tga
texture[94]     name: 'textures/sstic/1604401524'   path: game/textures/sstic/1604401524.tga
texture[96]     name: 'textures/sstic/1509983054'   path: game/textures/sstic/1509983054.tga
texture[97]     name: 'textures/sstic/31570422'     path: game/textures/sstic/31570422.tga
texture[122]    name: 'textures/sstic/86520831'     path: game/textures/sstic/86520831.tga
texture[123]    name: 'textures/sstic/838783866'    path: game/textures/sstic/838783866.tga
texture[138]    name: 'textures/sstic/71921642'     path: game/textures/sstic/71921642.tga
texture[193]    name: 'textures/sstic/1773100136'   path: game/textures/sstic/1773100136.tga
```

2.2.5 Bruteforcer la clef

Si on souhaitait bruteforcer la clef, il faut prendre en compte les 80 tuiles customs du répertoire, qui contiennent chacune 3 tronçons de hash, soit 240 tronçons au total. Comme on doit obtenir les 8 tronçons dans l'ordre (8 tronçons de 32 bits pour former une clef de 256 bits), il faut compter 240^8 essais, soit 11007531417600000000 (11 milliards de milliards), ce qui est clairement hors de portée. Cette méthode était donc à bannir.

2.3 Stage 3 - USB Paint

Nous pouvons commencer par copier le stage 3 dans un répertoire dédié, vérifier son intégrité et l'extraire :

```
$ mkdir -p stage3
$ cp stage3.zip stage3/
$ cd stage3
$ unzip -t stage3.zip
Archive:  stage3.zip
    testing: encrypted                OK
    testing: memo.txt                 OK
    testing: paint.cap                OK
No errors detected in compressed data of stage3.zip.
$ unzip stage3.zip
Archive:  stage3.zip
 extracting: encrypted
  inflating: memo.txt
  inflating: paint.cap
$ rm -f stage3.zip
$ file *
encrypted: data
memo.txt:  ASCII text
paint.cap: tcpdump capture file (little-endian) - version 2.4, capture length 262144)
```

Nous observons trois fichiers : de la donnée brute, un fichier texte et une capture de paquets réseau. Commençons par lire le fichier texte :

```
$ cat memo.txt
Cipher: Serpent-1-CBC-With-CTS
IV: 0x53535449433230313520d537461676533
Key: Well, definitely can't remember it... So this time I securely stored it with Paint.

SHA256: 6b39ac2220e703a48b3de1e8365d9075297c0750e9e4302fc3492f98bdf3a0b0 - encrypted
SHA256: 7beabe40888fbbf3f8ff8f4ee826bb371c596dd0cebe0796d2dae9f9868dd2d2 - decrypted
```

On comprend ici que le fichier encrypted est un fichier chiffré en Serpent-1 [10], avec le mode d'opération CBC (*Cipher Block Chaining* [11]) et avec la variante CTS (*CipherText Stealing* [12]) qui permet principalement d'éviter d'avoir recours au padding.

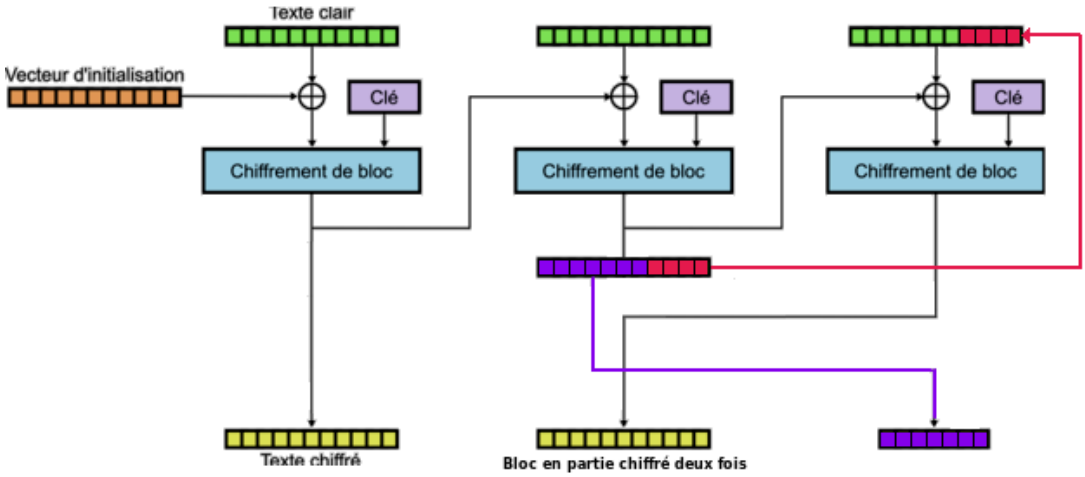


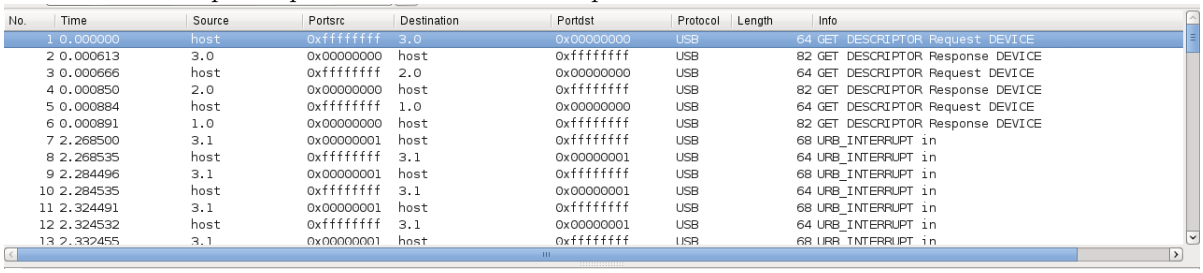
FIGURE 1 – Fonctionnement du mode d'opération CBC avec variante CTS

10. http://en.wikipedia.org/wiki/Serpent_(cipher)
11. http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_Block_Chaining_.28CFB.29
12. http://en.wikipedia.org/wiki/Ciphertext_stealing

On dispose également du hash SHA256 du fichier chiffré et de celui du fichier déchiffré.
Vérifions tout d'abord que le hash du fichier chiffré est correct :

```
$ S3_SHA256=6b39ac2220e703a48b3de1e8365d9075297c0750e9e4302fc3492f98bdf3a0b0
$ echo "$S3_SHA256 *encrypted" > SHA256SUMS
$ sha256sum -c SHA256SUMS
encrypted : OK
```

L'IV (Initialization Vector) est fourni, il ne manque que la clef, que l'on doit apparemment retrouver grâce à Paint, une application utilisée pour dessiner. Notre intuition nous amène à consulter les paquets stockés dans la capture que nous avons à notre disposition :

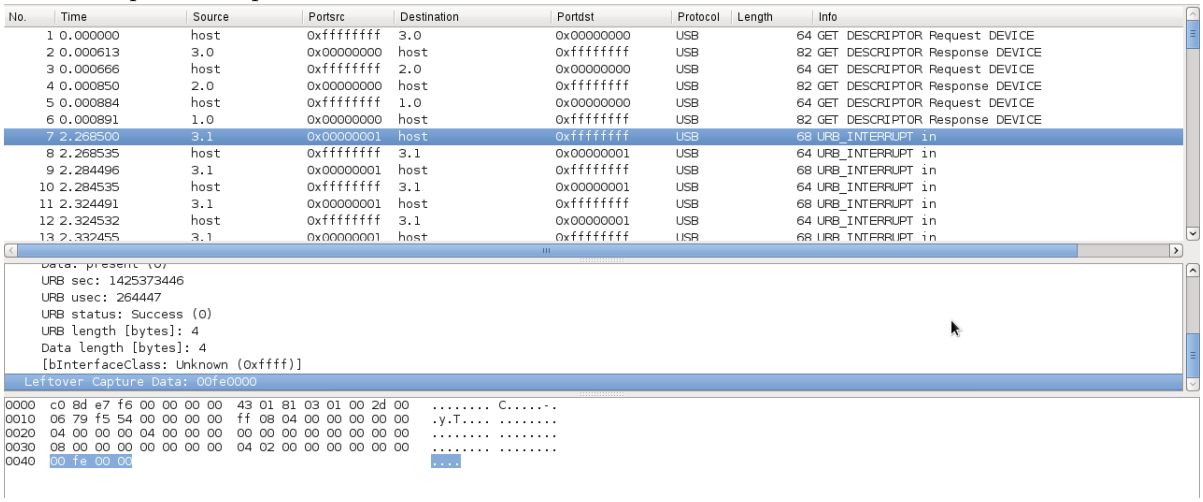


No.	Time	Source	Portsrc	Destination	Portdst	Protocol	Length	Info
1	0.000000	host	0xffffffff	3.0	0x00000000	USB	64	GET DESCRIPTOR Request DEVICE
2	0.000613	3.0	0x00000000	host	0xffffffff	USB	82	GET DESCRIPTOR Response DEVICE
3	0.000666	host	0xffffffff	2.0	0x00000000	USB	64	GET DESCRIPTOR Request DEVICE
4	0.000850	2.0	0x00000000	host	0xffffffff	USB	82	GET DESCRIPTOR Response DEVICE
5	0.000884	host	0xffffffff	1.0	0x00000000	USB	64	GET DESCRIPTOR Request DEVICE
6	0.000891	1.0	0x00000000	host	0xffffffff	USB	82	GET DESCRIPTOR Response DEVICE
7	2.268500	3.1	0x00000001	host	0xffffffff	USB	68	URB_INTERRUPT in
8	2.268535	host	0xffffffff	3.1	0x00000001	USB	64	URB_INTERRUPT in
9	2.284496	3.1	0x00000001	host	0xffffffff	USB	68	URB_INTERRUPT in
10	2.284535	host	0xffffffff	3.1	0x00000001	USB	64	URB_INTERRUPT in
11	2.324491	3.1	0x00000001	host	0xffffffff	USB	68	URB_INTERRUPT in
12	2.324532	host	0xffffffff	3.1	0x00000001	USB	64	URB_INTERRUPT in
13	2.332455	3.1	0x00000001	host	0xffffffff	USB	68	URB_INTERRUPT in

On remarque immédiatement qu'il s'agit d'une capture de communications USB. On voit que trois périphériques sont raccordés sur les ports 1, 2, et 3. Déterminons rapidement leur nature [13] :

Port	idVendor, idProduct	Device Description
3.0	0x04b3, 0x310c	IBM Corp. / Wheel Mouse
2.0	0x8087, 0x0024	Intel Corp. / Integrated Rate Matching Hub
1.0	0x1d6b, 0x0002	Linux Foundation / 2.0 root hub

En observant attentivement, on constate que seul le périphérique sur le port 3 dialogue avec la machine hôte par la suite. Concentrons-nous donc uniquement sur celui-ci. Il s'agit d'une souris à molette. L'ensemble des messages sont des messages d'interruption en entrée et des acquittements de la part de l'hôte. On peut voir que la taille de la couche de données est assez faible : 4 octets.



No.	Time	Source	Portsrc	Destination	Portdst	Protocol	Length	Info
1	0.000000	host	0xffffffff	3.0	0x00000000	USB	64	GET DESCRIPTOR Request DEVICE
2	0.000613	3.0	0x00000000	host	0xffffffff	USB	82	GET DESCRIPTOR Response DEVICE
3	0.000666	host	0xffffffff	2.0	0x00000000	USB	64	GET DESCRIPTOR Request DEVICE
4	0.000850	2.0	0x00000000	host	0xffffffff	USB	82	GET DESCRIPTOR Response DEVICE
5	0.000884	host	0xffffffff	1.0	0x00000000	USB	64	GET DESCRIPTOR Request DEVICE
6	0.000891	1.0	0x00000000	host	0xffffffff	USB	82	GET DESCRIPTOR Response DEVICE
7	2.268500	3.1	0x00000001	host	0xffffffff	USB	68	URB_INTERRUPT in
8	2.268535	host	0xffffffff	3.1	0x00000001	USB	64	URB_INTERRUPT in
9	2.284496	3.1	0x00000001	host	0xffffffff	USB	68	URB_INTERRUPT in
10	2.284535	host	0xffffffff	3.1	0x00000001	USB	64	URB_INTERRUPT in
11	2.324491	3.1	0x00000001	host	0xffffffff	USB	68	URB_INTERRUPT in
12	2.324532	host	0xffffffff	3.1	0x00000001	USB	64	URB_INTERRUPT in
13	2.332455	3.1	0x00000001	host	0xffffffff	USB	68	URB_INTERRUPT in

```
Data: present (0)
URB sec: 1425373446
URB usec: 264447
URB status: Success (0)
URB length [bytes]: 4
Data length [bytes]: 4
[bInterfaceClass: Unknown (0xffff)]
Leftover Capture Data: 00fe0000

0000  c0 8d e7 f6 00 00 00 00  43 01 81 03 01 00 2d 00   ........ C.....-.
0010  06 79 f5 54 00 00 00 00  ff 08 04 00 00 00 00 00   .y.T.... ........
0020  04 00 00 00 04 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0030  08 00 00 00 00 00 00 00  04 02 00 00 00 00 00 00   ........ ........
0040  00 fe 00 00                                       ....
```

En cherchant rapidement sur le net [14], il est possible de comprendre le sens de ces messages :

- 1 bit - bouton 1 *(clic gauche)*
- 1 bit - bouton 2
- 1 bit - bouton 3
- 1 bit - bouton 4
- 1 bit - bouton 5
- 3 bits - reservé
- 8 bits - dX *(différentiel de déplacement horizontal)*
- 8 bits - dY *(différentiel de déplacement vertical)*
- 8 bits - molette

13. `http://www.linux-usb.org/usb.ids`
14. `http://www.usbmadesimple.co.uk/ums_5.htm`

Il ne reste donc plus qu'à décoder tous ces messages. Comme l'utilisateur parle d'utiliser Paint, on imagine qu'il a enregistré son traffic USB pendant une session de dessin. Pour la reproduire, nous allons donc rejouer les messages enregistrés et appliquer les transformations sur une image blanche.

Afin d'automatiser ce processus, il était nécessaire de rédiger un petit script :

Listing 6 – src/stage3/usbreplay.py

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

"Stage3 USB Replay"

__version__ = '0.1'

import struct
import PIL.Image
import PIL.ImageDraw
from scapy.all import PcapReader, Raw

INF    = 10**9
MARGIN = 20

def dataprovider(filename):
    "parse pcap file and yield decoded data"
    for pkt in PcapReader(filename):
        data = pkt[Raw].load
        if data[9:10] == '\x02':
            continue
        data = pkt[Raw].load[0x40:]
        if not data:
            continue
        yield struct.unpack('Bbbb', data)

def main():
    "main function"
    # première lecture du flux: on détermine la zone de dessin
    x, y, minx, maxx, miny, maxy, p = 0, 0, INF, -INF, INF, -INF, False
    for _buttons, _dx, _dy, _wheel in dataprovider('paint.cap'):
        x, y, p    = x + _dx, y + _dy, _buttons == 1
        if p:
            minx, maxx = min(minx, x), max(maxx, x)
            miny, maxy = min(miny, y), max(maxy, y)
    offx = -minx + MARGIN
    offy = -miny + MARGIN
    sizx = maxx - minx + 2 * MARGIN
    sizy = maxy - miny + 2 * MARGIN
    # deuxième lecture du flux, on trace le dessin
    img = PIL.Image.new('1', (sizx, sizy), 'white')
    drw = PIL.ImageDraw.Draw(img)
    (x, y), printing = (offx, offy), False
    for _buttons, _dx, _dy, _wheel in dataprovider('paint.cap'):
        printing = _buttons == 1
        if printing:
            drw.line((x, y, x + _dx, y + _dy), 'black')
        x, y = x + _dx, y + _dy
    img.save('paint.png')
    img.show()

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

La taille de l'image et la position de la souris au début de l'enregistrement USB sont calculés en ajoutant une marge de largeur constante autour des positions minimum et maximum atteintes par le curseur pendant le tracé. Cela implique de lire deux fois le fichier de capture. On peut économiser une lecture si on détermine la taille et la position de la zone de dessin de manière empirique.

Et en le lançant, nous obtenons une image du replay USB à la souris :

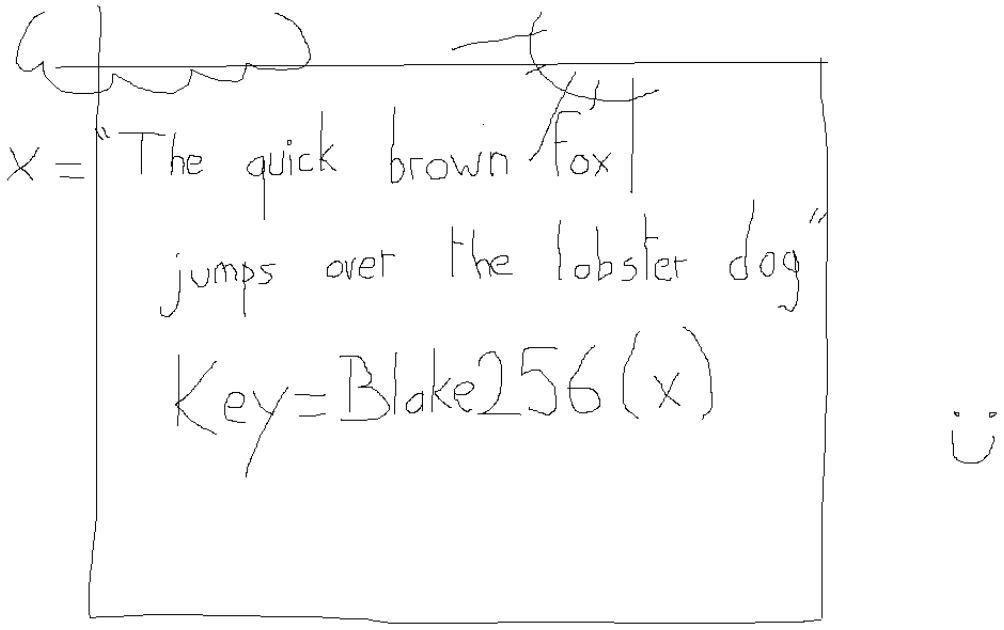


FIGURE 2 – src/stage3/paint.png

On retient donc que la clef est le hash Blake256 [15] du message :
`"The quick brown fox jumps over the lobster dog"`.

On notera un clin d'oeil au pangramme [16], qui est normalement "over the *lazy* dog".

Bref, il est temps de trouver une implémentation de Blake256. L'implémentation Python de l'auteur [17] suffisait amplement. En ce qui concerne Serpent-1, l'implémentation de référence est très très compliquée à utiliser : il faut lui ajouter la fonctionnalité des modes de chiffrement par blocs (qui n'est pas présente nativement), les opérations sur les tableaux d'octets sont délicates car l'outil fourni manipule une structure interne de tableau de bits qui sont dans l'ordre mais pour lesquels les octets, eux, sont inversés. Bref. Une implémentation à l'API beaucoup plus naturelle (proche de *Crypto.Cipher*) est heureusement disponible dans le module Python CryptoPlus [18]. C'est celle que nous allons utiliser.

Il ne reste maintenant plus qu'à déchiffrer notre donnée chiffrée. Pour l'automatiser, nous avons rédigé un petit script :

Listing 7 – src/stage3/decode.py

```
1 #!/usr/bin/env python
2 #-*- coding :utf-8 -*-
3
4 "Stage2 Serpent-1-CBC-CTS decoder"
5
```

15. http://en.wikipedia.org/wiki/BLAKE_(hash_function)
16. http://en.wikipedia.org/wiki/Pangram
17. http://www.seanet.com/~bugbee/crypto/blake/
18. https://github.com/doegox/python-cryptoplus

```
6  __version__ = '0.1'
7
8  import blake
9  import hashlib
10 from CryptoPlus.Cipher.python_Serpent import MODE_CBC, new
11
12 IV         = '535354494332303135 2d537461676533'.decode('hex')
13 SHA256_E = '6b39ac2220e703a48b3de1e8365d9075297c0750e9e4302fc3492f98bdf3a0b0'
14 SHA256_D = '7beabe40888fbbf3f8ff8f4ee826bb371c596dd0cebe0796d2dae9f9868dd2d2'
15
16 def main(txtkey):
17     "main function"
18     key = blake.BLAKE(256).hexdigest(txtkey).decode('hex')
19     with open('encrypted', 'rb') as encrypted:
20         data = encrypted.read()
21         assert(hashlib.sha256(data).hexdigest() == SHA256_E)
22         N  = len(data) / 16 + 1
23         _C = [None] + [data[16 * k:16 * (k + 1)] for k in xrange(N)]
24         # Astuce
25         Dn = new(key, MODE_CBC).decrypt(_C[N-1])
26         _C[N] += Dn[-2:]
27         _C[N-1], _C[N] = _C[N], _C[N-1]
28         # ######
29         data = ''.join(_C[i] for i in xrange(1, N + 1))
30         msg  = new(key, MODE_CBC, IV).decrypt(data)
31         msg  = msg[:-2]  # /!\ REMOVE PADDING BEFORE HASH !
32         if  hashlib.sha256(msg).hexdigest() == SHA256_D:
33             outfile = 'decrypted'
34             print 'OK', outfile
35             open(outfile, 'wb').write(msg)
36
37 if  __name__ == '__main__':
38     main('The quick brown fox jumps over the lobster dog')
```

Nous utilisons une petite astuce pour gérer correctement le CTS. On peut étendre le fichier chiffré afin que sa taille soit un multiple de la taille d'un bloc AES, et ainsi déchiffrer simplement avec CBC comme si de rien n'était [19]. C'est très pratique.

Si nous avions dû réimplémenter nous-même le mode de chiffrement par bloc CBC, une deuxième astuce pour accélérer énormément le processus de déchiffrement consiste à paralléliser les opérations de déchiffrement de chaque bloc.

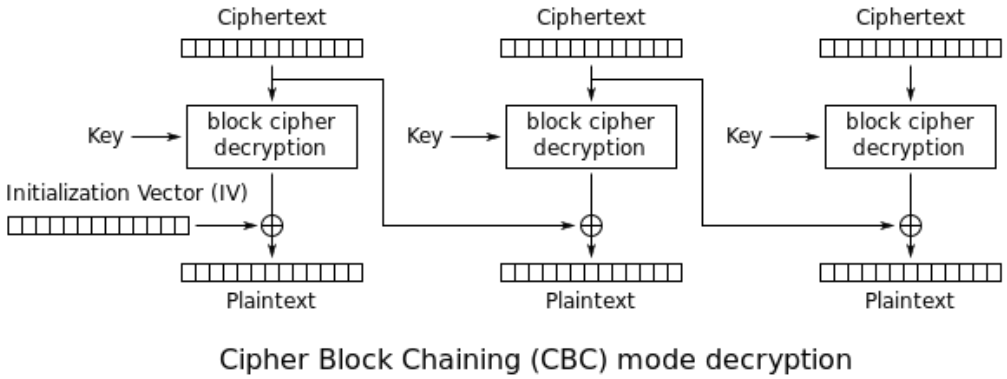


FIGURE 3 – Parallélisation du déchiffrement en mode d'opération CBC

En effet, en mode CBC, tous les blocs peuvent être déchiffrés en parallèle d'abord. Ensuite, pour obtenir un bloc en clair, il suffit d'appliquer un XOR dessus avec le bloc chiffré précédent (ou l'IV dans le cas du premier bloc). Cela permet de gagner un temps significatif sur des machines multi-processeurs.

19. http://en.wikipedia.org/wiki/Ciphertext_stealing#CBC_ciphertext_stealing_decryption_using_a_standard_CBC_interface

⚠ **Attention !** Les concepteurs ont choisi de calculer le hash **après** retrait du *padding* (ce qui peut valoir de longues minutes de debugging).

Comme précédemment, le fichier déchiffré est une archive ZIP qui constituera notre stage 4.

```
$ file decrypted
decrypted: Zip archive data, at least v2.0 to extract
$ cp decrypted ../stage4.zip
$ cd ..
```

2.4 Stage 4 - Désobscurcissement JS

Nous pouvons commencer par copier le stage 4 dans un répertoire dédié, vérifier son intégrité, et l'extraire :

```
$ mkdir -p stage4
$ cp stage4.zip stage4/
$ cd stage4
$ unzip -t stage4.zip
Archive:  stage4.zip
    testing: stage4.html              OK
No errors detected in compressed data of stage4.zip.
$ unzip stage4.zip
Archive:  stage4.zip
  inflating: stage4.html
$ rm -f stage4.zip
$ file *
stage4.html: HTML document, ASCII text, with very long lines, with CRLF line terminators
```

Un seul fichier, et il s'agit d'une page web. Regardons ce qu'elle contient.

```
<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 = "2b1f25cf8db5d243f59b065da6b56[...]b1944bfd3ac1646ffe2";
        var hash = "08c3be636f7dffd91971f65be4cec3c6d162cb1c";
        $=~[];$={___:++$,$$$$:(![]+"")[$],__$:++$,[...]+$.__$+"\\"+$.__$+$.__$+""))())();
    </script>
</body>
</html>
```

Il s'agit donc principalement de données probablement chiffrées (`data`), d'un hash (`hash`) et d'un code JavaScript obscurci (ou alors JavaScript c'est vraiment pire que dans mes souvenirs...). Instinctivement, on souhaite désobscurcir ce code JavaScript. Faisons cela en plusieurs étapes.

Après avoir extrait le code obscurci dans un fichier `stage4.0.js`, on le passe à `js-beautify`[20] qui donne ceci :

```
$ = ~[];
$ = {
    ___: ++$,                 # 0
    __$: ++$,                 # 1
    _$_: ++$,                 # 2
    _$$: ++$,                 # 3
    $___: ++$,                # 4
    $_$: ++$,                 # 5
    $$_: ++$,                 # 6
    $$$: ++$,                 # 7
```

20. `https://github.com/beautify-web/js-beautify`

```
    $___: ++$,                  # 8
    $__$: ++$,                  # 9
    $_$_: (![] + "")[$],        # undefined
    $_$$: ({} + "")[$],         # b
    $$__: ({} + "")[$],         # b
    $$_$: ($[$] + "")[$],       # undefined
    $$$_: (!"" + "")[$],        # undefined
    $$$$: (![] + "")[$],        # undefined
};
$.$_ = ($.$_ = $ + "")[$.$_$] + ($._$ = $.$_[$.__$]) + ($.$$ = ($.$ + "")[$.__$]) +
       ((!$) + "")[$._$$] + ($.__ = $.$_[$.$$_]) + ($.$ = (!"" + "")[$.__$]) +
       ($._ = (!"" + "")[$._$_]) + $.$_[$.$_$] + $.__ + $._$ + $.$;
$.$$ = $.$ + (!"" + "")[$._$$] + $.__ + $._ + $.$ + $.$$;
$.$ = ($.___)[$.$_][$.$_];
$.$($.$($.$$ + "\"" + "__=" + $.$$_$ + [...] + "\\" + $.__$ + $.__$ + "\"")())();
```

Nous avons raccourci le dernier bloc pour des raisons de lisibilité, mais il représente la quasi totalité du contenu. On reconnait la méthode d'obscurcissement appelée `jjencode` [21]. Il existe des décodeurs sur le net (notamment Decoder-JJEncode [22]), mais l'ignorant au moment de la résolution de cette partie du challenge, j'ai choisi d'effectuer tout le processus à la main. On remarque que ce code est constitué de deux `$.$(...)()` imbriqués. Il serait intéressant de savoir ce que représente la variable `$.$`. On modifie alors le code et on l'exécute en ligne de commande :

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

print("[DEBUG]" + $.$_);

$.$ = ($.___)[$.$_][$.$_];

print("[DEBUG]" + $.$);

$.$($.$($.$$ + "\"" + "__=" + $.$$_$ + [...] + "\\" + $.__$ + $.__$ + "\"")())();
```

```
$ js stage4.0.js
[DEBUG]constructor
[DEBUG]function Function() {
    [native code]
}
stage4.0b.js:24:0 ReferenceError: document is not defined
```

C'est un appel dynamique à un constructeur ! On va donc pouvoir exposer le code à l'intérieur pour l'afficher. On retire donc le double `$.$(...)()` imbriqué, que l'on remplace par un appel à `print`. Nous sauvegardons le tout dans un fichier `stage4.1.js`.

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

À ce stade, une simple exécution du script va commencer à remplacer une partie des variables par leur valeur effective, laissant apparaître une structure un peu plus familière. Le print que l'on a ajouté nous débarasse de toutes les problématiques de modules non chargés, ou plus généralement de conditions non satisfaites. On l'exécute donc :

```
$ js stage4.1.js > stage4.2.js
```

21. http://utf-8.jp/public/jjencode.html
22. https://github.com/jacobsoo/Decoder-JJEncode/blob/master/decoder.html

et on obtient :

```
return"
__=docu\155e\156t;
$$$='\163ta\147e5';
$$_$='load';
$_$$='\40';
_$$$$$='u\163e\162';
_$$$='d\151\166';
$$_$$$='\156a\166\151\147ato\162';
$$_$$='\160\162efe\162e\156ce\163';
$_$$$='to';
[...]
;}).catc\150(fu\156ct\151o\156(){__[____](___$)[__$___]=$;});
}$$[$_________](_____________________________,$_$);\12\11\11"
```

Il s'agit d'une fonction qui retourne une chaîne de caractères qui contient du code (assurément puisque, rappelons-nous, nous sommes dans un contexte de construction dynamique de fonction). Pour analyser ce code plus rapidement, nous allons retirer le `return`, remplacer les caractères qui sont encodés en octal par leur représentation ASCII, et l'exécuter.

```
$ js stage4.2.js > stage4.3.js
```

et on obtient :

Listing 8 – src/stage4/stage4.3.js

```
1  __ = document;
2  $$$ = 'stage5';
3  $$_$ = 'load';
4  $_$$ = ' ';
5  _$$$$$ = 'user';
6  _$$$ = 'div';
7  $$_$$$ = 'navigator';
8  $$_$$ = 'preferences';
9  $_$$$ = 'to';
10 $$$_$ = 'href';
11 $$$$_ = '=';
12 $$$$$ = 'chrome';
13 _$$$$ = '"';
14 $_$$$$ = 'Agent';
15 $$$_$$ = 'down';
16 $$$$_$ = 'import';
17 $ = '<b>Failed' + $_$$ + $_$$$ + $_$$ + $$_$ + $_$$ + $$$ + '</b>';
18 ____ = 'write';
19 _____ = 'getElementById';
20 _$_ = 'raw';
21 $$ = window;
22 __$ = $$.crypto.subtle;
23 __$_ = 'decrypt';
24 ___$ = 'status';
25 $_____ = $$$$_$ + 'Key';
26 ________ = 0;
27 __$__ = 'then';
28 _$_____ = 'digest';
29 __$____ = 'innerHTML';
30 ___$__ = {
31     name: 'SHA-1'
32 };
33 _____$_ = data;
34 ______$ = hash;
35 _$______ = Blob;
36 ___$____ = URL;
37 _____$__ = 'createObjectURL';
38 _______$ = 'type';
39 $________ = 'application/octet-stream';
40 _$_______ = 'name';
41 __$______ = 'AES-CBC';
42 ___$_____ = 'iv';
43 _____$____ = '<a' + $_$$ + $$$_$ + $$$$_ + _$$$$;
```

```
44  _____$__ = '"' + $_$$ + $$$_$$ + $$_$ + $$$$_ + _$$$$ + $$$ + '.zip' + _$$$$ + '>' + $ [...]
45  _______$_ = '(';
46  ________$ = ')';
47  $_________ = 'setTimeout';
48  __________ = parseInt;
49  __________ = $$[$$_$$$][_$$$$$ + $_$$$$];
50  ___________ = 'length';
51  ____________ = 'substr';
52  _____________ = _______ + 1;
53  ______________ = _____________ * 2;
54  _______________ = _____________ * 4;
55  ________________ = _____________ * _____________;
56  $_$ = 125 * _________________;
57  _________________ = 'indexOf';
58  __________________ = 'charCodeAt';
59  ___________________ = 'push';
60  ____________________ = Uint8Array;
61  _____________________ = '';
62  ______________________ = 'byteLength';
63  _______________________ = $_$$$ + 'String';
64  __[___]('<h' + _____________ + '>Down' + $$_$ + $_$$ + 'manager</h' + _____________ [...]
65  __[___]('<' + _$$$ + $_$$ + 'id' + $$$$_ + _$$$$ + ___$ + _$$$$ + '><i>' + $$_$ + 'ing [...]
66  __[___]('<' + _$$$ + $_$$ + 'style' + $$$$_ + _$$$$ + 'display:none' + _$$$$ + '><a' + [...]
67
68  function _____(______) {
69      _ = [];
70      for (___________ = _______; ___________ < ______[___________]; ++___________) _[_ [...]
71      return new ____________________(_);
72  }
73
74  function ____________________(______) {
75      _ = [];
76      for (___________ = _______; ___________ < ______[___________] / _____________; [...]
77      return new ____________________(_);
78  }
79
80  function ____________________(________) {
81      _______ = ____________________;
82      for (___________ = _______; ___________ < ________[____________________]; ++_ [...]
83          ___ = ________[___________][____________________](_____________________);
84          if (___[___________] < _____________) _______ += _______;
85          _______ += ___;
86      }
87      return _______;
88  }
89
90  function ____________________() {
91      $_ = _____(__________[_____________](__________[____________________](______$_) + __ [...]
92      _$___ = _____(__________[_____________](__________[____________________](_______$) - [...]
93      _$ = {};
94      _$[_$_______] = __$_____;
95      _$[___$_____] = $_;
96      _$[_____________] = _$__[_____________] * _________________;
97      __$[$_____](_$_, _$___, _$, false, [__$_])[__$__](function(_$___) {
98          __$[__$_](_$, _$___, ____________________(_____$_))[__$__](function(___$_) {
99              ____$ = new ____________________(___$_);
100             __$[_$____](___$__, _____$)[__$__](function(_____$$) {
101                 if (_____$ == ____________________(new ____________________(_____$ [...]
102                     _____$_ = {};
103                     _____$_[_______$] = $________;
104                     _____$ = new _$_____([_____$], _____$_);
105                     __$_____ = ___$____[_____$__](_____$);
106                     __[_____](___$)[__$___] = ____$___ + __$_____ + _____$__;
107                 } else {
108                     __[_____](___$)[__$___] = $;
109                 }
110             });
111         }).catch(function() {
112             __[_____](___$)[__$___] = $;
113         });
```

```
    }).catch(function() {
        __[_____](____$)[__$____] = $;
    });
}
$$[$_________](_____________________________, $_$);
```

Voilà qui est beaucoup plus intéressant ! Grâce à un autre script Python, nous pouvons remplacer les variables déclarées en préambule par leur valeur réelle, ce qui donne :

Listing 9 – src/stage4/stage4.4.js

```
___$__ = {
    name: 'SHA-1'
};
__[ 'write' ]('<h' + 1 + '>Down' + 'load' + ' ' + 'manager</h' + 1 + '>');
__[ 'write' ]('<' + 'div' + ' ' + 'id' + '=' + '"' + 'status' + '"' + '><i [...]
__[ 'write' ]('<' + 'div' + ' ' + 'style' + '=' + '"' + 'display:none' + '"' [...]

function _____ ( ______ ) {
    _ = [];
    for ( i = 0 ; i < ______ [ 'length' ]; ++i ) _ [ 'push' ]( ______ [ 'ch [...]
    return new Uint8Array ( _ );
}

function ____________________ ( ______ ) {
    _ = [];
    for ( i = 0 ; i < ______ [ 'length' ] / 2 ; ++i ) _ [ 'push' ]( parseInt [...]
    return new Uint8Array ( _ );
}

function ____________________ ( ________ ) {
    ______ = '';
    for ( i = 0 ; i < ________ [ 'byteLength' ]; ++i ) {
        'write' = ________ [ i ][ 'toString' ]( 16 );
        if ( 'write' [ 'length' ] < 2 ) ______ += 0 ;
        ______ += 'write' ;
    }
    return ______ ;
}

function ____________________ () {
    $_ = _____ ( window [ 'navigator' ][ 'userAgent' ] [ 'substr' ]( window [ 'na [...]
    _$__ = _____ ( window [ 'navigator' ][ 'userAgent' ] [ 'substr' ]( window [ ' [...]
    _$ = {};
    _$ [ 'name' ] = 'AES-CBC' ;
    _$ [ 'iv' ] = $_ ;
    _$ [ 'length' ] = _$__ [ 'length' ] * 8 ;
    window.crypto.subtle [ 'importKey' ]( 'raw' , _$__ , _$ , false, [ 'decrypt' [...]
        window.crypto.subtle [ 'decrypt' ]( _$ , _$___ , ____________________ ( [...]
            _____$ = new Uint8Array ( ___$_ );
            window.crypto.subtle [ 'digest' ]( ___$__ , _____$ )[ 'then' ](function [...]
                if ( hash == ____________________ (new Uint8Array ( _____$$ ))) {
                    ______$_ = {};
                    ______$_ [ 'type' ] = 'application/octet-stream' ;
                    hash = new Blob ([ _____$ ], ______$_ );
                    __$_____ = URL [ 'createObjectURL' ]( hash );
                    document [ 'getElementById' ]( 'status' )[ 'innerHTML' ] = '<a h [...]
                } else {
                    document [ 'getElementById' ]( 'status' )[ 'innerHTML' ] = '<b>F [...]
                }
            });
        }).catch(function() {
            document [ 'getElementById' ]( 'status' )[ 'innerHTML' ] = '<b>Failed to [...]
        });
    }).catch(function() {
        document [ 'getElementById' ]( 'status' )[ 'innerHTML' ] = '<b>Failed to loa [...]
    });
}
$$[ 'setTimeout' ]( ____________________ , 1000 );
```

Les dernières variables ont définitivement perdu leur nom, mais en analysant le code et en le réécrivant en partie, on peut finalement arriver (par un processus entièrement manuel cette fois) à obtenir ce genre de code :

Listing 10 – src/stage4/stage4.5.js

```
1  document.write('<h1>Download manager</h1>');
2  document.write('<div id="status"><i>loading...</i></div>');
3  document.write('<div style="display:none"><a target="blank" href="chrome://browser/con [...]
4
5  function toascii(input) {
6      result = [];
7      for (i = 0; i < input.length; ++i)
8          result.push(input.charCodeAt(i));
9      return new Uint8Array(result);
10 }
11
12 function hextoint(input) {
13     result = [];
14     for (i = 0; i < input.length / 2; ++i)
15         result.push(parseInt(input.substr(i * 2, 2), 16));
16     return new Uint8Array(result);
17 }
18
19 function tohex(input) {
20     result = '';
21     for (i = 0; i < input.byteLength; ++i) {
22         hexa = input[i].toString(16);
23         if (hexa.length < 2)
24             result += 0;
25         result += hexa;
26     }
27     return result;
28 }
29
30 function mainloop() {
31     iv   =  toascii(navigator.userAgent.substr(navigator.userAgent.indexOf('(') +  1, 16));
32     key  =  toascii(navigator.userAgent.substr(navigator.userAgent.indexOf(')') - 16, 16));
33     ctx  = {name: 'AES-CBC', iv: iv, length: key.length * 8};
34     window.crypto.subtle.importKey('raw', key, ctx, false, ['decrypt']).then(
35         function(result_importKey) {
36             window.crypto.subtle.decrypt(ctx, result_importKey, hextoint(data)).then(
37                 function(result_decrypt) {
38                     result_data = new Uint8Array(result_decrypt);
39                     window.crypto.subtle.digest({name: 'SHA-1'}, result_data).then(
40                         function(result_digest) {
41                             if (hash == tohex(new Uint8Array(result_digest))) {
42                                 flow = {};
43                                 flow.type = 'application/octet-stream';
44                                 h    = new Blob([result_data], flow);
45                                 url  = URL.createObjectURL(h);
46                                 document.getElementById('status').innerHTML = '<a href= [...]
47                             } else {
48                                 document.getElementById('status').innerHTML = '<b>Faile [...]
49                             }
50                         });
51                 }).catch(function() {
52                     document.getElementById('status').innerHTML = '<b>Failed to load s [...]
53                 });
54         }).catch(function() {
55             document.getElementById('status').innerHTML = '<b>Failed to load stage5</b>';
56         });
57 }
58
59 $$.setTimeout(mainloop, 1000);
```

À la ligne 31, on remarque que l'IV est constitué des 16 **premiers** caractères qui **suivent** la **première** parenthèse **ouvrante** dans le *UserAgent* du navigateur qui sert à consulter la page. À la ligne 32, on remarque que la clef est constituée des 16 **derniers** caractères qui **précèdent** la **première** parenthèse

fermante dans le *UserAgent* du navigateur qui sert à consulter la page. À la ligne 33, on remarque que l'algorithme de chiffrement est AES-CBC (couramment utilisé). Et à la ligne 39, on remarque que le hash est un digest SHA-1.

On dispose désormais de toutes les informations pour déchiffrer notre bloc de données présent dans `stage4.html`, sauf du *UserAgent* du navigateur. La première idée était de le bruteforcer par dictionnaire. J'ai utilisé une base de données personnelle (non fournie). Le résultat ne s'est pas fait attendre.

Il était également possible de le bruteforcer directement, en composant un *UserAgent* crédible. Des indices étaient donnés :

- la police *Lucida Grande* (dans le CSS) a été utilisée dans l'interface de Mac OS X de 1999 à 2014, avant d'être remplacée par *Helvetica Neue* dans Max OS X 10.10. Elle n'est habituellement pas présente sous Windows ni sous Linux, donc on peut conjecturer que le *UserAgent* est spécifique à Mac OS X ;
- le lien vers `chrome://browser/content/preferences/preferences.xul` est spécifique à *Firefox* pour les versions supérieures à 1.5 (le nom du fichier pour les préférences était `pref/pref.xul` jusqu'alors). On peut donc aussi conjecturer que le *UserAgent* est spécifique à une version récente de *Firefox* ;
- on pouvait également vérifier la compatibilité des différents appels à l'API JavaScript en fonction du support par les navigateurs, sur les sites web qui répertorient la documentation. On constate par exemple que `URL.createObjectURL()` n'est disponible sur *Firefox* qu'à partir de la version 4.0.

Il était probablement possible de réduire encore le champs des possibilités.

On pouvait alors tester tous les *UserAgents* qui correspondaient à l'expression régulière suivante : `(Macintosh; Intel Mac OS X \d{2}\.\d; rv:\d{2}.\d)`

Pour automatiser le bruteforce, nous avons rédigé un script :

Listing 11 – src/stage4/decode.py

```
1  #!/usr/bin/env python
2  #-*- coding:utf-8 -*-
3
4  "Stage3 AES-CBC decoder"
5
6  __version__ = '0.1'
7
8  import re
9  import sys
10 import hashlib
11 import gzip
12 import Crypto.Cipher.AES
13
14 def main():
15     "main function"
16     with open('stage4.html', 'r') as stage4:
17         tmp     = stage4.read().splitlines()
18         data    = re.search(r'"(.*?)"', tmp[11]).group(1).decode('hex')
19         h4sh    = re.search(r'"(.*?)"', tmp[12]).group(1)
20     with gzip.open('UA-LIST.txt.gz', 'r') as ualist:
21         lines = ualist.read().splitlines()
22         n     = len(lines)
23         for i in xrange(n):
24             ua  = lines[i]
25             if  '(' not in ua or ')' not in ua:
26                 continue
27             iv  = ua[ua.index('(') +  1:][:16]
28             if  len(iv)  != 16:
29                 continue
30             key = ua[ua.index(')') - 16:][:16]
31             if  len(key) != 16:
32                 continue
33             ctx = Crypto.Cipher.AES.new(key, Crypto.Cipher.AES.MODE_CBC, iv)
34             msg = ctx.decrypt(data)
35             msg = msg.rstrip('\x0C')   # /!\ REMOVE PADDING BEFORE h4sh !
36             print >> sys.stderr, '%d/%d %s' % (i, n, ua)
37             if  hashlib.sha1(msg).hexdigest() == h4sh:
38                 outfile = 'decrypted'
39                 print 'OK', outfile
```

```
40                  open(outfile, 'wb').write(msg)
41                  return 0
42      return 1
43
44  if __name__ == '__main__':
45      exit(main())
```

⚠ **Attention !** Les concepteurs ont choisi de calculer le hash **après** retrait du *padding* (ce qui peut valoir de longues minutes de debugging).

Et en le lançant :

```
$ python decode.py
[...]
66475/116990 Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:31.0) Gecko/20100101 Firefox/31.0
66476/116990 Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:32.0) Gecko/20100101 Firefox/32.0
66477/116990 Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:33.0) Gecko/20100101 Firefox/33.0
66478/116990 Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:34.0) Gecko/20100101 Firefox/34.0
66480/116990 Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:35.0) Gecko/20100101 Firefox/35.0
OK decrypted
```

On obtient un *UserAgent* valide (en effet, étant donné que seule la sous-chaîne entre parenthèses compte, il existe plusieurs *UserAgents* qui permettent de déchiffrer notre donnée chiffrée) : `Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:35.0) Gecko/20100101 Firefox/35.0`.

Comme précédemment, le fichier déchiffré est une archive ZIP qui constituera notre stage 5.

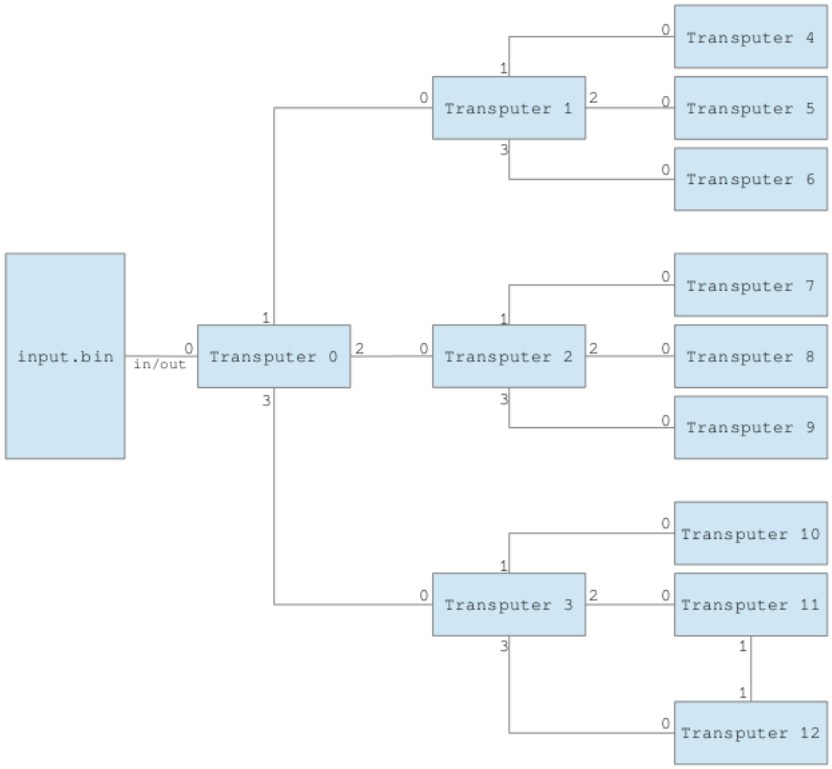
```
$ file decrypted
decrypted: Zip archive data, at least v2.0 to extract
$ cp decrypted ../stage5.zip
$ cd ..
```

2.5 Stage 5 - Reverse ST20

Nous pouvons commencer par copier le stage 5 dans un répertoire dédié, vérifier son intégrité et l'extraire :

```
$ mkdir -p stage5
$ cp stage5.zip stage5/
$ cd stage5
$ unzip -t stage5.zip
Archive:  stage5.zip
    testing: input.bin                OK
    testing: schematic.pdf            OK
No errors detected in compressed data of stage5.zip.
$ unzip stage5.zip
Archive:  stage5.zip
  inflating: input.bin
  inflating: schematic.pdf
$ rm -f stage5.zip
$ file *
input.bin:     data
schematic.pdf: PDF document, version 1.4
```

Il s'agit donc d'un document PDF et d'un binaire. Ouvrons prudemment le PDF dans un viewer à jour :



On y trouve aussi une indication intéressante :

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

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

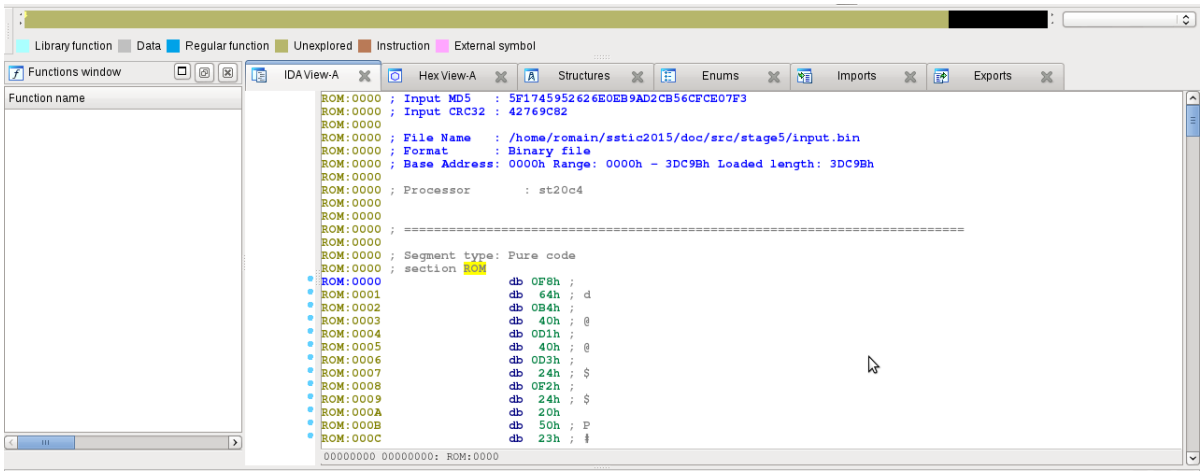
L'indication cruciale, c'est que les concepteurs du challenge font du Python ! Plus sérieusement, on voit ici un schéma avec les canaux de communications entre plusieurs transputers [23], le hash SHA256 de la donnée chiffrée, celui de la donnée déchiffrée, un vecteur de test (ce qui indique qu'il va probablement falloir implémenter soi-même la routine de déchiffrement car elle n'est pas standard) et un indice très important : on parle ici d'une architecture ST20 [24].

Pour l'étape de reverse-engineering, plusieurs possibilités, mais nous n'allons en décrire que deux : soit on travaille avec IDA [25] (en statique), soit on travaille avec Miasm2 [26] (en dynamique). Les deux sont possibles. Pour Miasm2, il faut implémenter toute l'architecture car elle n'est pas fournie en standard. J'ai joint un ensemble de patch files en annexe de ce document, qui sont basé sur le commit `9dad900cba94d1ca9aa095bbfe57927e2c3f46d4`.

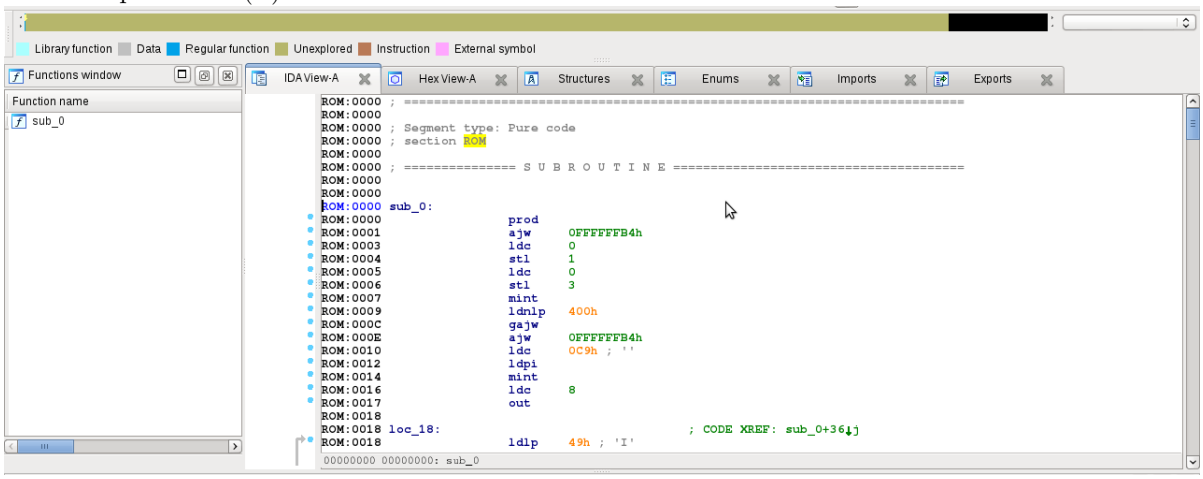
2.5.1 Avec IDA Free

Dans IDA, il est possible de désassembler les opcodes (il faut choisir l'architecture "SGS-Thomson ST20 : st20c4" pour avoir accès au bon jeu d'instructions, on s'en rend compte très rapidement - si on choisit le st20 classique, IDA ne désassemble que quelques octets et s'arrête).

23. http://en.wikipedia.org/wiki/Transputer
24. http://pdf.datasheetcatalog.com/datasheet/SGSThomsonMicroelectronics/mXruvtu.pdf
25. https://www.hex-rays.com/products/ida/support/download_freeware.shtml
26. https://github.com/cea-sec/miasm



Visiblement, comme les opcodes ne sont pas enveloppés dans un binaire structuré, IDA qui ne trouve pas d'entry point ne commence le désassemblage nulle part. On va donc l'aider un peu en tentant de créer une procédure (P) à l'adresse 0x0000.



Cela semble fonctionner, et la fonction se termine en 0x00DB. Puis en fouillant un peu, on remarque des chaînes de caractères ASCII (A), notamment en 0x00DD (`'Boot ok\x00'`), 0x00E5 (`'Code ok\x00'`), 0x00ED (`'Decrypt\x00'`), 0x0985 (`'KEY:'`), et 0x996 (`'congratulations.tar.bz2'`).

Si on cherche à faire l'inventaire des fonctions (un peu à l'intuition, en fonction de la tête des octets et des prologues st20c4), en fonction de leur adresse :

- 0x0000 ;
- 0x0105, 0x0182 et 0x01FF qui sont assez semblables ;
- la série des 9 fonctions 0x0288, 0x02C5, 0x0302, 0x033F, 0x037C, 0x03B9, 0x03F6, 0x0433, 0x0470 sont formellement identiques ;
- 0x04C5 (qui CALL dans 0x04B9, 0x04BF) ;
- 0x052D (qui CALL dans 0x0521, 0x0527) ;
- 0x0595 (qui CALL dans 0x0589, 0x058F) ;
- 0x0639 (qui CALL dans 0x062D, 0x0633) ;
- 0x06B5 (qui CALL dans 0x06A9, 0x06AF) ;
- 0x0769 (qui CALL dans 0x075D, 0x0763) ;
- 0x07D5 (qui CALL dans 0x07C9, 0x07CF) ;
- 0x0885 (qui CALL dans 0x0879, 0x087F) ;
- 0x090D (qui CALL dans 0x0901, 0x0907) ;
- après 0x09AD, tout semble être du garbage, car si on tente de désassembler, de nombreuses erreurs de jumps et d'alignement apparaîssent.

On dispose alors globalement de 22 fonctions, les 4 au début, les 9 identiques ensuite, et les 9 dernières avec leur deux sous-routines respectives. L'intuition nous fait dire que les 9 fonctions identiques ne servent pas (pourquoi dupliquer du code inutilement ?). Donc de prime abord, nous nous concentrons sur les 13

autres, et puis nous reviendrons dessus si finalement les autres ne sont pas suffisantes.

Nous allons donc réduire notre première passe de reverse-engineering à 13 fonctions. Comme le nombre de transputers sur le schéma... coïncidence ?! Je ne pense pas. De là à supposer que chacun des transputers se voit attribuer l'exécution d'une partie du code égale aux fonctions mises en évidence, il n'y a qu'un pas. On va supposer que les transputers sont déclarés dans l'ordre.

Entre les fonctions, on remarque les offsets, channels et longueurs qui servent au mécanisme de *Peek and Poke*[27] des différents transputers. Mais on ne s'attardera pas dessus. En effet, on suppose au début que les fonctions sont plaquées dans l'ordre. Et il s'est avéré que la supposition était bonne ; je ne suis donc pas revenu sur ces opérations mémoire par la suite.

Commençons par reverser les fonctions que l'on attribue aux transputers les plus profonds (ceux qui ont le moins d'entrées/sorties). On remarquera que pour chacun, leur exécution est faite en boucle, qu'ils prennent en entrée un buffer de 12 octets et renvoient 1 octet à chaque tour.

Fonction 0x04C5 :

```
sub_4C5 :
    ajw     0FFFFFFFBh      ; allocation de place dans la pile
    ldc     0
    stl     1
    ldc     0               ; \
    ldlp    1               ;  |-- accumulateur initialisé à 0
    sb                      ; /

loc_4CD :                   ; début de la boucle principale
    ldc     0Ch             ; \
    stl     0               ; / len = 0xC
    ldlp    2               ; buffer
    mint                    ; \
    ldnlp   4               ; / channel 0 : 0x80000010
    ldl     6
    call    sub_4C5_IN      ; IN : réception de 0xC octets
    ldc     0
    stl     0               ; index courant

loc_4D8 :
    ldl     0               ; \
    ldlp    2               ;  |-- *(buffer + index)
    bsub                    ;  |    = buffer[index]
    lb                      ; /
    ldlp    1               ; valeur de l'accumulateur
    lb
    bsub                    ; ajout
    ldc     0FFh            ; \
    and                     ; /  on ne garde que le LSB
    ldlp    1
    sb                      ; on stocke dans l'accumulateur
    ldl     0               ; \
    adc     1               ;  |-- on incrémente l'index
    stl     0               ; /
    ldc     0Ch             ; \
    ldl     0               ;  |-- test de fin de chaîne
    gt                      ; /
    cj      loc_4EF
    j       loc_4D8

loc_4EF :
    ldc     1               ; \
    stl     0               ; / len = 0x1
    ldlp    1               ; accumulateur
    mint                    ; channel 0 : 0x80000000
    ldl     6
    call    sub_4C5_OUT     ; OUT
    j       loc_4CD         ; boucle après l'initialisation
```

Pour résumer, si on devait écrire une fonction de transfert pour ce transputer :

27. http://en.wikipedia.org/wiki/PEEK_and_POKE

```
static accumulateur = 0
while true :
    buffer = INPUT(12)
    for index = 0 to 11 :
        accumulateur += buffer[index]
        accumulateur &= 0xFF
    OUTPUT(accumulateur)
```

Fonction 0x052D :

On remarque que cette fonction est relativement similaire à la précédente. La seule opération qui change est le BSUB qui devient un XOR.

```
sub_52D :
    ajw     0FFFFFFFBh      ; allocation de place dans la pile
    ldc     0
    stl     1
    ldc     0               ; \
    ldlp    1               ;  |-- accumulateur initialisé à 0
    sb                      ; /

loc_535 :                   ; début de la boucle principale
    ldc     0Ch             ; \
    stl     0               ; / len = 0xC
    ldlp    2               ; buffer
    mint                    ; \
    ldnlp   4               ; / channel 0 : 0x80000010
    ldl     6
    call    sub_52D_IN      ; IN : réception de 0xC octets
    ldc     0
    stl     0               ; index courant

loc_540 :
    ldl     0               ; \
    ldlp    2               ;  |-- *(buffer + index)
    bsub                    ;  |    = buffer[index]
    lb                      ; /
    ldl     1               ; valeur de l'accumulateur
    xor                     ; ou exclusif
    ldc     0FFh            ; \
    and                     ; /  on ne garde que le LSB
    ldlp    1
    sb                      ; on stocke dans l'accumulateur
    ldl     0               ; \
    adc     1               ;  |-- on incrémente l'index
    stl     0               ; /
    ldc     0Ch             ; \
    ldl     0               ;  |-- test de fin de chaîne
    gt                      ; /
    cj      loc_557
    j       loc_540

loc_557 :
    ldc     1               ; \
    stl     0               ; / len = 0x1
    ldlp    1               ; accumulateur
    mint                    ; channel 0 : 0x80000000
    ldl     6
    call    sub_52D_OUT     ; OUT
    j       loc_535         ; boucle après l'initialisation
```

Pour résumer, si on devait écrire une fonction de transfert pour ce transputer :

```
static accumulateur = 0
while true :
    buffer = INPUT(12)
    for index = 0 to 11 :
        accumulateur ^= buffer[index]
        accumulateur &= 0xFF
    OUTPUT(accumulateur)
```

Fonction 0x0595 :

Ce transputer est un peu plus complexe. Dans la boucle de calcul, on peut analyser les formules comme si elles étaient écrites en notation polonaise inversée.

```
sub_595 :
    ajw     0FFFFFFF9h      ; allocation de place dans la pile
    ldc     0
    stl     2
    ldc     0               ; \
    stl     1               ; / initialisation de l'accumulateur
    ldc     0               ; \
    stl     3               ; / variable d'état initialisée à 0

loc_59D :                   ; début de la boucle principale
    ldc     0Ch             ; \
    stl     0               ; / len = 0xC
    ldlp    4               ; buffer
    mint                    ; \
    ldnlp   4               ; / channel 0 : 0x80000010
    ldl     8
    call    sub_595_IN      ; IN : réception de 0xC octets
    ldl     3               ; \
    eqc     0               ;  |-- vérification de l'état
    cj      loc_5C5         ; /
    ldc     0
    stl     0               ; index courant

loc_5AC :
    ldl     0               ; \
    ldlp    4               ;  |-- *(buffer + index)
    bsub                    ;  |   = buffer[index]
    lb                      ; /
    ldl     1               ; valeur de l'accumulateur
    bsub                    ; ajout
    ldc     0FFFFh          ; \
    and                     ; /  on ne garde que les 2 LSB
    stl     1               ; on stocke dans l'accumulateur
    ldl     0               ; \
    adc     1               ;  |-- on incrémente l'index
    stl     0               ; /
    ldc     0Ch             ; \
    ldl     0               ;  |-- test de fin de chaîne
    gt                      ; /
    cj      loc_5C3
    adc     0
    j       loc_5AC

loc_5C3 :
    ldc     1               ; \
    stl     3               ; / l'état est passé à 1

loc_5C5 :
    ldl     1               ; v       \
    ldc     8000h           ; 0x8000   |
    and                     ; and      |-- (v & 0x8000) >> 0xF
    ldc     0Fh             ; 0x0F     |
    shr                     ; shr     /

    ldl     1               ; v       \
    ldc     4000h           ; 0x4000   |
    and                     ; and      |-- (v & 0x4000) >> 0xE
    ldc     0Eh             ; 0x0E     |
    shr                     ; shr     /

    xor                     ; ou exclusif entre les deux

    ldc     0FFFFh          ; \
    and                     ; / on ne garde que les 2 LSB

    ldl     1               ; v      \
```

```
    ldc     1               ; 1         |
    shl                     ; shl       |-- (v << 1) & 0xFFFF
    ldc     0FFFFh          ; 0xFFFF    |
    and                     ; and       /

    xor                     ; ou exclusif entre les deux

    ldc     0FFFFh          ; \
    and                     ; / on ne garde que les 2 LSB

    dup
    stl     1               ; on stocke l'accumulateur
    ldc     0FFh            ; \
    and                     ;  |
    ldlp    2               ;  |-- on garde le LSB pour le OUT
    sb                      ; /
    ldc     1
    stl     0
    ldlp    2
    mint                    ; channel 0 : 0x80000000
    ldl     8
    call    sub_595_OUT     ; OUT
    j       loc_59D         ; boucle après l'initialisation
```

Pour résumer, si on devait écrire une fonction de transfert pour ce transputer :

```
static state = 0
static accumulateur = 0
while true :
    buffer = INPUT(12)
    if  state == 0 :
        for index = 0 to 11 :
            accumulateur += buffer[index]
            accumulateur &= 0xFFFF
        state = 1
    accumulateur = ((accumulateur & 0x8000) >> 0xF) ^
                   ((accumulateur & 0x4000) >> 0xE) ^
                   ((accumulateur << 0x1) & 0xFFFF)
    OUTPUT(accumulateur & 0xFF)
```

Fonction 0x0639 :

Maintenant que nous en avons fait trois, nous pouvons accélérer un peu sur les explications. On ne recopiera pas non plus tout le code de la fonction, mais seulement la partie responsable du calcul effectif des valeurs en entrées / sorties.

```
loc_64C :
    ldl     0
    ldlp    4
    bsub
    lb                      ; buffer[index]
    ldl     1               ; accumulateur1
    bsub                    ;     +
    ldc     0FFh            ;     0xFF
    and                     ;        &
    stl     1               ; et on le restocke
    ldlp    4
    ldl     0
    adc     6               ; on décale l'index
    bsub
    lb                      ; buffer[index+6]
    ldl     2               ; accumulateur2
    bsub                    ;     +
    ldc     0FFh            ;     0xFF
    and                     ;        &
    stl     2               ; et on le restocke
    ldl     0
    adc     1
    stl     0
    ldc     6
    ldl     0
```

```
    gt                      ; test de fin de boucle
    cj      loc_66C
    j       loc_64C

loc_66C :
    ldl     2               ; accumulateur2
    ldl     1               ; accumulateur1
    xor                     ;      ^
    ldc     0FFh            ;      0xFF
    and                     ;          &
    ldlp    3
    sb                      ; et on le stocke
    ldc     1
    stl     0
    ldlp    3
    mint
    ldl     8
    call    sub_639_OUT
    j       loc_63D
```

Pour résumer, si on devait écrire une fonction de transfert pour ce transputer :

```
while true :
    accumulateur1, accumulateur2 = 0, 0
    buffer = INPUT(12)
    for index = 0 to 6 :
        accumulateur1 += buffer[index]
        accumulateur1 &= 0xFF
        accumulateur2 += buffer[index+6]
        accumulateur2 &= 0xFF
    OUTPUT(accumulateur1 ^ accumulateur2)
```

Fonction 0x06B5 :

La première partie du code effectue une somme sur les octets passés en paramètre. On ne s'attardera pas dessus. La suite en revanche est plus intéressante. On remarque que le résultat de la somme est placé dans une case différente à chaque fois, ou plus précisément dans un ring de taille 4. À ce stade, les quatres valeurs en mémoire sont XOR-ées, et l'index de position dans le ring est incrémenté.

```
loc_6FD :
    ldl     2               ;
    ldc     3               ;
    prod                    ;
    ldlp    5               ;
    wsub                    ;
    ldl     0               ;
    bsub                    ; lecture d'un caractère
    lb                      ; dans le ring courant
    ldl     1               ;
    bsub                    ; somme avec l'accumulateur
    ldc     0FFh            ;
    and                     ; et on ne garde que le LSB
    stl     1
    ldl     0               ; \
    adc     1               ;  |-- on incrémente l'index
    stl     0               ; /
    ldc     0Ch
    ldl     0
    gt                      ; on teste la fin du buffer
    cj      loc_716
    adc     0
    j       loc_6FD

loc_716 :
    ldl     3               ; résultat final
    ldl     1               ; accmulateur
    xor                     ; XOR
    ldc     0FFh
    and                     ; puis on ne garde que le LSB
    ldlp    3
```

```
    sb
    ldl     2               ; case courante dans le ring
    adc     1               ; on passe à la case suivante
    stl     2               ; on stocke
    ldc     4               ; \
    ldl     2               ;  |-- test des 4 cases du ring
    gt                      ; /
    cj      loc_72B
    adc     0
    j       loc_6F9

loc_72B :
    ldc     1
    stl     0
    ldlp    3
    mint
    ldl     12h
    call    sub_6B5_OUT     ; puis on envoie le résultat
    j       loc_6DB
```

Pour résumer, si on devait écrire une fonction de transfert pour ce transputer :

```
ringindex = 0
while true :
    ring[ringindex] = 0
    buffer = INPUT(12)
    for index = 0 to 11 :
        ring[ringindex] += buffer[index]
        ring[ringindex] &= 0xFF
    OUTPUT(ring[0] ^ ring[1] ^ ring[2] ^ ring[3])
    ringindex = (ringindex + 1) % 3
```

Fonction 0x0769 :

Ce transputer est très très simple :

```
loc_77C :
    ldl     0
    ldlp    2
    bsub
    lb                      ; on lit le caractère
    ldl     0               ; index
    ldc     7               ; 7
    and                     ; index & 7
    shl                     ; buffer[index] << (index & 7)
    ldl     1               ; accumulateur
    xor                     ; XOR
    ldc     0FFh            ; 0xFF
    and                     ; AND
    ldlp    1
    sb                      ; on stocke ici le résultat
    ldl     0
    adc     1
    stl     0
    ldc     0Ch
    ldl     0
    gt                      ; test de fin de buffer
    cj      loc_799
    j       loc_77C

loc_799 :
    ldc     1
    stl     0
    ldlp    1
    mint
    ldl     6
    call    sub_769_OUT     ; puis on envoie le résultat
    j       loc_76D
```

Pour résumer, si on devait écrire une fonction de transfert pour ce transputer :

```
while true :
    accumulateur = 0
    buffer = INPUT(12)
    for index = 0 to 11 :
        accumulateur ^= buffer[index] << (index & 0x7)
        accumulateur &= 0xFF
    OUTPUT(accumulateur)
```

Fonction 0x07D5 :

Comme pour la fonction en 0x06B5, cette fonction conserve un ring des 4 derniers bytearray en entrée. Un accumulateur temporaire calcule la somme des premiers octets de chaque entrée de ring. L'index dans le ring est incrémenté de la même manière que précédemment.

```
loc_817 :
    ldl     0
    ldc     3
    prod
    ldlp    4
    wsub
    lb                      ; premier octet
    ldl     1               ; accumulateur
    bsub                    ; +
    ldc     0FFh            ; 0xFF
    and                     ; &
    stl     1
    ldl     0
    adc     1               ; incrément de l'index du ring
    stl     0
    ldc     4
    ldl     0
    gt                      ; test si fin de ring (4)
    cj      loc_82E
    adc     0
    j       loc_817

loc_82E :
    ldl     1               ; accumulateur
    ldc     3               ; 3
    and                     ; accumulateur & 3
    ldc     3               ;
    prod                    ;
    ldlp    4               ;
    wsub                    ; calcul de l'offset du ring

    ldl     1               ; accumulateur
    ldc     4               ; 4
    shr                     ; accumulateur >> 4
    ldc     0Ch             ; 12
    rem                     ; (accumulateur >> 4) % 12
    ldc     0FFh            ; 0xFF
    and                     ; (accumulateur >> 4) % 12
    bsub                    ; qu'on ajoute à l'offset du ring

    lb                      ; ring[acc & 3][acc >> 4 % 12]
    ldlp    3
    sb
    ldc     1
    stl     0
    ldlp    3
    mint
    ldl     11h
    call    sub_7D5_OUT     ; puis on envoie le résultat
    j       loc_7FB
```

Pour résumer, si on devait écrire une fonction de transfert pour ce transputer :

```
ringindex = 0
while true :
    accumulateur = 0
```

```
    buffer = INPUT(12)
    ring[ringindex] = buffer
    for i in [0, 1, 2, 3]:
        accumulateur += ring[i][0]
        accumulateur &= 0xFF
    OUTPUT(ring[accumulateur & 3][(accumulateurtmp >> 4) % 12])
    ringindex = (ringindex + 1) % 3
```

Les deux fonctions suivantes inter-agissent, mais comme les autres transputers, elles peuvent être modélisées par une seule et même fonction à état. On notera ici qu'il est obligatoire (mais du coup plus simple) de combiner les deux transputers dans une même fonction à état.

```
static state = 0
while true:
    buffer = INPUT(12)
    OUTPUT_11(buffer[state % 12])
    OUTPUT_12(buffer[(buffer[0] ^ buffer[3] ^ buffer[7]) % 0xC])
    state = (buffer[1] ^ buffer[5] ^ buffer[9]) & 0xFF
```

Les trois transputers 1, 2, et 3 sont identiques. Inutile de montrer une fois de plus leur code, il est relativement simple :

- ils recoivent 0xC octets via le canal 0 ;
- ils le transmettent aux transputers sous-jaçents ;
- ils réceptionnent 1 octet de chaque transputers ;
- et ils transmettent par le canal 0 le XOR des trois valeurs.

Enfin, en ce qui concerne le transputer 0 (aussi appelé transputer racine), on peut décrire son fonctionnement (qui traduit le processus de déchiffrement d'un octet de données) en pseudo-langage :

```
while buffer is not empty:
    output the character
    ALn    = input from link n (for n in 1, 2, 3)
    AL1   ^= AL2 ^ AL3
    D      = C ^ (i + 2 * KEY[i])
    KEY[i] = AL1
    i++
    output D on link 0
```

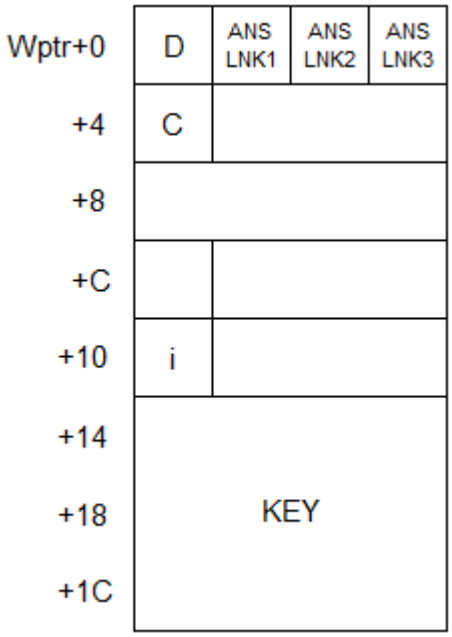


FIGURE 4 – Mapping mémoire du fonctionnement du transputer racine

Toutes ces fonctions ont donc un équivalent relativement simple en pseudo-language. Il est donc temps de les implémenter dans un script pour tester leur validité sur le vecteur de test fourni dans le PDF.

Listing 12 – src/stage5/testvector.py

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

import sys
import hashlib

class Transputer:
    def __init__(self):
        self.reset()
    def reset(self):
        self.value = 0x00
        self.state = 0x00
        self.sring = ['\x00'*12] * 4
    def test(self, expect):
        result = ''
        self.send('\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C')
        result += self.recv()
        self.send('\x92\x03\xE0\x99\xBA\x57\x93\x7A\x2B\x9D\x7C\x14')
        result += self.recv()
        self.send('\xDE\x05\x48\x82\x18\x13\x0B\x89\xBB\xE0\x88\x7E')
        result += self.recv()
        self.send('\x8C\x86\x7D\x7B\x09\xC1\x48\x1A\xE1\x8F\x06\x46')
        result += self.recv()
        self.send('\x6A\x06\x0B\x9E\x9E\xFE\x0E\x41\x65\x3F\x14\x60')
        result += self.recv()
        if result != expect:
            print >> sys.stderr, '[DEBUG]', '%r != %r' % (result, expect)
        assert(result == expect)
        print >> sys.stderr, self.__class__.__name__, 'test OK!'
        self.reset()
    def recv(self):
        if isinstance(self.value, (list,)):
            return ''.join(map(lambda x: chr(x & 0xFF), self.value))
        return chr(self.value & 0xFF)

class Transputer4(Transputer):
    def send(self, data):
        assert(len(data) == 0xC)
        for c in map(ord, data):
            self.value = (self.value + c) & 0xFF

class Transputer5(Transputer):
    def send(self, data):
        assert(len(data) == 0xC)
        for c in map(ord, data):
            self.value = (self.value ^ c) & 0xFF

class Transputer6(Transputer):
    def send(self, data):
        assert(len(data) == 0xC)
        if self.state == 0x00:
            for c in map(ord, data):
                self.value = (self.value + c) & 0xFFFF
        self.value = (((self.value & 0x8000) >> 0xF) ^
                      ((self.value & 0x4000) >> 0xE) ^
                      ((self.value << 0x1) & 0xFFFF))
        self.state = 0x01

class Transputer7(Transputer):
    def send(self, data):
        assert(len(data) == 0xC)
        value1, value2 = 0, 0
        for c in map(ord, data[:6]):
            value1 = (value1 + c) & 0xFF
        for c in map(ord, data[6:]):
```

```
66              value2 = (value2 + c) & 0xFF
67          self.value = (value1 ^ value2) & 0xFF
68
69  class Transputer8(Transputer):
70      def send(self, data):
71          assert(len(data) == 0xC)
72          self.sring[self.state] = data
73          self.value = 0
74          for i in xrange(4):
75              self.value ^= sum(map(ord, self.sring[i]))
76          self.state = (self.state + 1) % 4
77
78  class Transputer9(Transputer):
79      def send(self, data):
80          assert(len(data) == 0xC)
81          self.value = 0
82          for i in xrange(12):
83              self.value = (self.value ^ (ord(data[i]) << (i & 0x7))) & 0xFF
84
85  class Transputer10(Transputer):
86      def send(self, data):
87          assert(len(data) == 0xC)
88          self.sring[self.state] = data
89          self.value = sum(ord(self.sring[x][0]) for x in xrange(4)) & 0xFF
90          self.value = ord(self.sring[self.value & 3][(self.value >> 4) % 0xC])
91          self.state = (self.state + 1) % 4
92
93  class Transputer1112(Transputer):
94      def send(self, data):
95          assert(len(data) == 0xC)
96          self.value = [ord(data[self.state % 0xC]),
97                        ord(data[(ord(data[0]) ^ ord(data[3]) ^ ord(data[7])) % 0xC])]
98          self.state = ord(data[1]) ^ ord(data[5]) ^ ord(data[9])
99          return self.value
100
101 def Transputer1(data):
102     assert(len(data) == 0xC)
103     T4.send(data)
104     T5.send(data)
105     T6.send(data)
106     t4v = ord(T4.recv())
107     t5v = ord(T5.recv())
108     t6v = ord(T6.recv())
109     return chr(t4v ^ t5v ^ t6v)
110
111 def Transputer2(data):
112     assert(len(data) == 0xC)
113     T7.send(data)
114     T8.send(data)
115     T9.send(data)
116     t7v = ord(T7.recv())
117     t8v = ord(T8.recv())
118     t9v = ord(T9.recv())
119     return chr(t7v ^ t8v ^ t9v)
120
121 def Transputer3(data):
122     assert(len(data) == 0xC)
123     T10.send(data)
124     T1112.send(data)
125     t10v       = ord(T10.recv())
126     t11v, t12v = map(ord, T1112.recv())
127     return chr(t10v ^ t11v ^ t12v)
128
129
130 T4 = Transputer4()
131 T5 = Transputer5()
132 T6 = Transputer6()
133 T7 = Transputer7()
134 T8 = Transputer8()
135 T9 = Transputer9()
```

```
136 T10 = Transputer10()
137 T1112 = Transputer1112()
138
139
140 def resetall():
141     T4.reset()
142     T5.reset()
143     T6.reset()
144     T7.reset()
145     T8.reset()
146     T9.reset()
147     T10.reset()
148     T1112.reset()
149
150 def decrypt(key, data):
151     result = ''
152     for i in xrange(len(data)):
153         AL1, AL2, AL3 = ord(Transputer1(key)), ord(Transputer2(key)), ord(Transputer3(key))
154         AL1 = AL1 ^ AL2 ^ AL3
155         j   = i % 0xC
156         X   = ord(data[i]) ^ ((j + 2 * ord(key[j])) & 0xFF)
157         key = key[:j] + chr(AL1) + key[j+1:]
158         result += chr(X)
159     return result
160
161
162 key    = '*SSTIC-2015*'
163 data   = '1d87c4c4e0ee40383c59447f23798d9fefe74fb82480766e'.decode('hex')
164 resetall()
165 result = decrypt(key, data)
166 assert(result == 'I love ST20 architecture')
167
168 print 'OK'
```

Et on l'exécute :

```
$ ./testvector.py
OK
```

Bien ! Passons donc maintenant au déchiffrement de la donnée chiffrée proprement dite. Déjà, il va falloir la trouver, l'extraire, et trouver la clef. Si on se remémore bien, nous avions trouvé une zone qui ne semblait pas être du code dans le fichier `input.bin` (elle démarrait en 0x09AD). Extrayons-la dans un fichier `encrypted` et vérifions si son hash SHA256 correspond bien à celui attendu.

```
$ python -c "open('encrypted', 'wb').write(open('input.bin', 'rb').read()[0x9AD:])"
$ S5_SHA256=a5790b4427bc13e4f4e9f524c684809ce96cd2f724e29d94dc999ec25e166a81
$ echo "$S5_SHA256 *encrypted" > SHA256SUMS
$ sha256sum -c SHA256SUMS
encrypted: OK
```

Parfait ! Là on a eu un peu de chance (mais on aurait aussi pu bruteforcer l'offset, la donnée était forcément fournie avec). Il ne reste maintenant plus qu'à trouver la clef de 12 octets.

```
$ hexdump -C input.bin | grep '000009[789a]'
00000970  77 66 94 20 65 0e 20 20  20 00 00 00 00 00 00 00  |wf. e.   ........|
00000980  00 00 00 00 00 4b 45 59  3a ff ff ff ff ff ff ff  |.....KEY:........|
00000990  ff ff ff ff ff 17 63 6f  6e 67 72 61 74 75 6c 61  |......congratula|
000009a0  74 69 6f 6e 73 2e 74 61  72 2e 62 7a 32 fe f3 50  |tions.tar.bz2..P|
```

Le contenu du fichier sous-entend que la clef est composée uniquement de 0xFF, mais le hash du fichier déchiffré n'est alors pas le bon. Du coup, il va falloir bruteforcer la clef. Dispose-t-on d'indices ? Oui. On s'attend à trouver un fichier au format BZip2 [28]. Et c'est un énorme indice. Y'a-t-il des particularités à un fichier bz2 qui peuvent restreindre notre espace de bruteforce ?

Voici un récapitulatif des différents champs présents dans l'entête :

```
.magic :16                          = 'BZ' signature/magic number
```

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

```
.version :8                          = 'h' for Bzip2 ('H'uffman coding), '0' for Bzip1 (deprecated)
.hundred_k_blocksize :8              = '1'..'9' block-size 100 kB-900 kB (uncompressed)
.compressed_magic :48                = 0x314159265359 (BCD (pi))
.crc :32                             = checksum for this block
.randomised :1                       = 0=>normal, 1=>randomised (deprecated)
.origPtr :24                         = starting pointer into BWT for after untransform
.huffman_used_map :16                = bitmap of ranges of 16 bytes, present/not present
.huffman_used_bitmaps :0..256        = bitmap of symbols used, present/not present (mult. of 16)
.huffman_groups :3                   = 2..6 number of different Huffman tables in use
.selectors_used :15                  = number of times that the Huffman tables are swapped
*.selector_list :1..6                = zero-terminated bit runs (0..62) of MTF'ed Huffman table
.start_huffman_length :5             = 0..20 starting bit length for Huffman deltas
*.delta_bit_length :1..40            = 0=>next symbol; 1=>alter length
                                                { 1=>decrement length;  0=>increment length }
.contents :2..inf                    = Huffman encoded data stream until end of block
.eos_magic :48                       = 0x177245385090 (BCD sqrt(pi))
.crc :32                             = checksum for whole stream
.padding :0..7                       = align to whole byte
```

Dans cet entête, donc :

- les 16 premiers bits d'un fichier bz2 sont 'BZ' ;
- les 8 bits suivants sont soit 'h' (bz2), soit '0' (bz1 - deprecated) ;
- les 8 bits suivants sont dans '1'...'9' ('9' par défaut) ;
- les 48 bits suivants valent 0x314159265359 (pi en BCD) ;

Sur une clef de 96 bits, les 80 premiers bits du texte clair sont presque fixes (du moins on peut le considérer dans un premier temps). Quel est l'espace de la clef ?

Sur le premier bloc, les octets de la clef sont utilisés conformément à cette formule :

`CLAIR = ( CHIFFRE ^ ( i + 2 * CLEF ) ) & 0xFF`

En fonction du chiffré donné et du clair attendu, on en déduit donc les possibilités pour la clef :

```
CHIFFRE = map(ord, '\xFE\xF3\x50\xDC\x81\xBC\x97\x27\x89\xAC')
CLAIR   = map(ord, 'BZh9\x31\x41\x59\x26\x53\x59')
for i in xrange(10):
    print ['0x%02X' % k for k in xrange(256)
    if CLAIR[i] == (CHIFFRE[i] ^ (i + 2 * k)) & 0xFF], \
    'pour que le %d-ieme octet 0x%02X devienne 0x%02X' % (
    i, CHIFFRE[i], CLAIR[i])

>>> ['0x5E', '0xDE'] pour que le 0-ieme octet 0xFE devienne 0x42
>>> ['0x54', '0xD4'] pour que le 1-ieme octet 0xF3 devienne 0x5A
>>> ['0x1B', '0x9B'] pour que le 2-ieme octet 0x50 devienne 0x68
>>> ['0x71', '0xF1'] pour que le 3-ieme octet 0xDC devienne 0x39
>>> ['0x56', '0xD6'] pour que le 4-ieme octet 0x81 devienne 0x31
>>> ['0x7C', '0xFC'] pour que le 5-ieme octet 0xBC devienne 0x41
>>> ['0x64', '0xE4'] pour que le 6-ieme octet 0x97 devienne 0x59
>>> ['0x7D', '0xFD'] pour que le 7-ieme octet 0x27 devienne 0x26
>>> ['0x69', '0xE9'] pour que le 8-ieme octet 0x89 devienne 0x53
>>> ['0x76', '0xF6'] pour que le 9-ieme octet 0xAC devienne 0x59
```

On a donc un espace de $10*1+2*8 = 26$ bits, ce qui est encore beaucoup, surtout si on doit déchiffrer la totalité du message, et vérifier le hash SHA256. Il y a une technique approximative mais efficace pour déterminer si une clef est certainement non-candidate (ou potentiellement candidate, avec une certaine probabilité).

Dans l'entête de BZip2, la map du codage de huffman contient plus de 0xFF qu'un flot de donnée aléatoire. Et l'avantage de cette table est qu'elle se trouve dans les 40 premiers octets du fichier en clair, donc il n'est pas nécessaire de déchiffrer la totalité du fichier. Si dans cette table, après avoir déchiffré les premiers octets, on trouve suffisamment de 0xFF, la clef est un bon candidat. Et on peut poursuivre sur les tests plus coûteux.

Une heuristique supplémentaire était encore envisageable avant de calculer le SHA256. Un autre magic number est situé en fin de fichier bz2 (la racine carré de PI en BCD). Mais deux inconvénient : le premier c'est qu'il n'est pas aligné sur un octet, donc les bits peuvent être difficiles à extraire (8 décalages possibles, de 0 bit à 7 bit) ; le second est que comme il est situé à la fin du fichier, on ne peut alors pas faire l'économie de déchiffrer tout le fichier en entrée.

Pour résumer, voici donc la méthode de bruteforce retenue avec les heuristiques appliquées :

1. on choisit les bits sûrs avec l'entête BZip2 ;
2. on choisit les deux bytes libres sur une valeur de boucle ;
3. on déchiffre les 40 premiers octets ;
4. on vérifie le nombre de 0xFF dans la map de l'encodage de huffman ;
5. si la clef est un bon candidat, on déchiffre en entier et on vérifie le hash ;
6. sinon on continue.

Une implémentation Python serait trop lente (après plusieurs tentatives). Il est donc plus judicieux ici de rédiger une version en C pour gagner en performances. Voici l'implémentation proposée :

Listing 13 – src/stage5/bruteforce.c

```c
/*
gcc -Wall -Wno-pointer-sign -pedantic -O2 -o bruteforce bruteforce.c -lssl -lcrypto
*/

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
#include <openssl/sha.h>

#define FILELEN 250606

typedef unsigned char uchar_t;

int EOS_MAGIC_BITS[48] = {
    0,0,0,1,0,1,1,1,
    0,1,1,1,0,0,1,0,
    0,1,0,0,0,1,0,1,
    0,0,1,1,1,0,0,0,
    0,1,0,1,0,0,0,0,
    1,0,0,1,0,0,0,0,
};
int eos[88];

uint32_t T4_value = 0;
uint32_t T5_value = 0;
uint32_t T6_state = 0;
uint32_t T6_value = 0;
uint32_t T8_index = 0;
uchar_t  T8_ring[4];
uint32_t T10_index = 0;
uchar_t  T10_ring[4][12];
uint32_t T1112_state = 0;

void Transputer123(uchar_t* data, uchar_t* out1, uchar_t* out2, uchar_t* out3)
{
    register int i; uint32_t v7l = 0, v7r = 0, v8 = 0, v9 = 0, v10 = 0;

    for (i = 0; i < 12; i++)
    {
        T4_value = (T4_value + data[i]) & 0xFF;
        T5_value = (T5_value ^ data[i]) & 0xFF;
    }
    if (!T6_state)
        for (i = 0; i < 12; i++)
            T6_value = (T6_value + data[i]) & 0xFFFF;
    T6_value = ((T6_value & 0x8000) >> 0xF) ^
               ((T6_value & 0x4000) >> 0xE) ^
               ((T6_value << 0x1) & 0xFFFF);
    T6_state = 1;
    for (i = 0; i < 6; i++)
    {
        v7l = (v7l + data[i])   & 0xFF;
```

```
57          v7r = (v7r + data[i+6]) & 0xFF;
58      }
59      T8_ring[T8_index] = 0;
60      for (i = 0; i < 12; i++)
61          T8_ring[T8_index] = (T8_ring[T8_index] + data[i]) & 0xFF;
62      for (i = 0; i < 4; i++)
63          v8 = (v8 ^ T8_ring[i]);
64      T8_index = (T8_index + 1) & 0x3;
65      for (i = 0; i < 12; i++)
66          v9 = (v9 ^ (data[i] << (i & 0x7))) & 0xFF;
67      for (i = 0; i < 12; i++)
68          T10_ring[T10_index][i] = data[i];
69      for (i = 0; i < 4; i++)
70          v10 = (v10 + T10_ring[i][0]) & 0xFF;
71      T10_index = (T10_index + 1) & 0x3;
72
73      *out1 = ((T4_value) ^ (T5_value) ^ ((T6_value) & 0xFF)) & 0xFF;
74      *out2 = (((v7l ^ v7r) & 0xFF) ^ ((v8) & 0xFF) ^ ((v9) & 0xFF)) & 0xFF;
75      *out3 = (((T10_ring[v10 & 3][(v10 >> 4) % 0xC]) & 0xFF) ^
76               (data[T1112_state % 0xC]) ^
77               (data[(data[0] ^ data[3] ^ data[7]) % 0xC])) & 0xFF;
78
79      T1112_state = (data[1] ^ data[5] ^ data[9]) & 0xFF;
80  }
81
82  void reset()
83  {
84      register int i, k;
85      T4_value = 0;
86      T5_value = 0;
87      T6_state = 0;
88      T6_value = 0;
89      T8_index = 0;
90      T10_index = 0;
91      T1112_state = 0;
92      for (k = 0; k < 4; k++)
93      {
94          T8_ring[k] = 0;
95          for (i = 0; i < 12; i++)
96              T10_ring[k][i] = 0;
97      }
98      bzero(eos, 88);
99  }
100
101 int decrypt(uchar_t* key, int len, uchar_t* data, uchar_t* out)
102 {
103     register int i, j; uchar_t AL1, AL2, AL3; int score; /* int x, fit; */
104     for (i = 0; i < len; i++)
105     {
106         Transputer123(key, &AL1, &AL2, &AL3);
107         AL1 = AL1 ^ AL2 ^ AL3;
108         j   = i % 0xC;
109         out[i] = data[i] ^ ((j + 2 * key[j]) & 0xFF);
110         key[j] = AL1;
111     }
112
113     score = 0;
114     for (i = 17; i < 17 + 16; i++) score += (out[i] == 0xFF) ? 1 : 0;
115     if  (score >= 10) return 0; else return 1;
116
117 /*
118     if  (out[0] != 'B') return 1;
119     if  (out[1] != 'Z') return 1;
120     if  (out[2] != '0' && data[2] != 'h')   return 1;
121     if  (!('1' <= out[3] && out[3] <= '9')) return 1;
122
123     for (i = 0; i < 11; i++)
124     {
125         x = out[len - 1 - i];
126         for (j = 0; j < 8; j++)
```

```
127             {
128                 eos[(10 - i) * 8 + 7 - j] = x & 1;
129                 x >>= 1;
130             }
131         }
132         for (i = 0; i < 8; i++)
133         {
134             fit = 0;
135             for (j = 0; j < 48; j++)
136                 if (eos[i + 1 + j] != EOS_MAGIC_BITS[j])
137                 {
138                     fit = 1;
139                     break;
140                 }
141             if (fit == 0)
142                 break;
143         }
144         return fit;
145 */
146 }
147
148 int main(int argc, char** argv)
149 {
150 #ifdef DEBUG
151     uchar_t tkey[12], tresult[24];
152     memcpy(tkey, "*SSTIC-2015*", 12);
153     reset();
154     decrypt(tkey, 24, "\x1D\x87\xC4\xC4\xE0\xEE\x40\x38\x3C\x59\x44\x7F"
155                       "\x23\x79\x8D\x9F\xEF\xE7\x4F\xB8\x24\x80\x76\x6E", tresult);
156     printf("%s\n", tresult);
157     exit(0);
158 #endif
159
160     int fd, i, flag, bz2, key[12];
161     uchar_t _key[12], data[FILELEN], result[FILELEN], hash[SHA256_DIGEST_LENGTH];
162     SHA256_CTX sha256;
163     fd = open(argv[1], O_RDONLY);
164     read(fd, data, FILELEN);
165     close(fd);
166
167     SHA256_Init(&sha256);
168     SHA256_Update(&sha256, data, FILELEN);
169     SHA256_Final(hash, &sha256);
170     assert(!memcmp(hash,
171         "\xA5\x79\x0B\x44\x27\xBC\x13\xE4\xF4\xE9\xF5\x24\xC6\x84\x80\x9C"
172         "\xE9\x6C\xD2\xF7\x24\xE2\x9D\x94\xDC\x99\x9E\xC2\x5E\x16\x6A\x81",
173         SHA256_DIGEST_LENGTH));
174
175     for (key[0] = 0x5E; key[0] <= 0xDE; key[0] += 0xDE - 0x5E)
176     for (key[1] = 0x54; key[1] <= 0xD4; key[1] += 0xD4 - 0x54)
177     for (key[2] = 0x1B; key[2] <= 0x9B; key[2] += 0x9B - 0x1B)
178     for (key[3] = 0x71; key[3] <= 0xF1; key[3] += 0xF1 - 0x71)
179     for (key[4] = 0x56; key[4] <= 0xD6; key[4] += 0xD6 - 0x56)
180     for (key[5] = 0x7C; key[5] <= 0xFC; key[5] += 0xFC - 0x7C)
181     for (key[6] = 0x64; key[6] <= 0xE4; key[6] += 0xE4 - 0x64)
182     for (key[7] = 0x7D; key[7] <= 0xFD; key[7] += 0xFD - 0x7D)
183     for (key[8] = 0x69; key[8] <= 0xE9; key[8] += 0xE9 - 0x69)
184     for (key[9] = 0x76; key[9] <= 0xF6; key[9] += 0xF6 - 0x76)
185     {
186     flag = 1;
187     for (key[10] = 0x00; key[10] <= 0xFF; key[10] ++)
188     for (key[11] = 0x00; key[11] <= 0xFF; key[11] ++)
189     {
190         reset();
191         for (i = 0; i < 12; i++) _key[i] = key[i] & 0xFF;
192
193         bz2 = decrypt(_key, 40, data, result);
194
195         if (!bz2 || flag)
196         {
```

```
197             fprintf(stderr, "trying ");
198             for (i = 0; i < 12; i++)
199                 fprintf(stderr, "%02X", key[i]);
200             fprintf(stderr, " ... ");
201             fprintf(stderr, "%d ", bz2);
202         }
203         if (bz2) {
204             if (flag) { fprintf(stderr, "\n"); flag = 0; }
205             continue;
206         }
207
208         fprintf(stderr, "possibly found?\n");
209
210         reset();
211         for (i = 0; i < 12; i++) _key[i] = key[i] & 0xFF;
212
213         decrypt(_key, FILELEN, data, result);
214
215         SHA256_Init(&sha256);
216         SHA256_Update(&sha256, result, FILELEN);
217         SHA256_Final(hash, &sha256);
218
219         for (i = 0; i < SHA256_DIGEST_LENGTH; i++)
220         fprintf(stderr, "%02X", hash[i]);
221         fprintf(stderr, " ");
222
223         if (!memcmp(hash, "\x91\x28\x13\x51\x29\xD2\xBE\x65\x28\x09\xF5\xA1\xD3\x37\x21\x1A"
224                           "\xFF\xAD\x91\xED\x58\x27\x47\x4B\xF9\xBD\x7E\x28\x5E\xCE\xF3\x21",
225             SHA256_DIGEST_LENGTH))
226         {
227             fprintf(stderr, "\n\033[32mfound!\033[m\n");
228             return 0;
229         }
230     }
231     }
232     fprintf(stderr, "\n\033[31mfailed.\033[m\n");
233
234     return 1;
235 }
```

Et on le lance :

```
$ gcc -Wall -Wno-pointer-sign -pedantic -O2 -o bruteforce bruteforce.c -lssl -lcrypto
$ ./bruteforce encrypted
trying 5ED49B7156FC64FD69F60000 ... 1
trying 5ED49B7156FC64FDE9760000 ... 1
trying 5ED49B7156FC64FDE9F60000 ... 1
trying 5ED49B7156FCE47D69760000 ... 1
trying 5ED49B7156FCE47D69F60000 ... 1
trying 5ED49B7156FCE47DE9760000 ... 1
trying 5ED49B7156FCE47DE976DAC5 ... 0 possibly found?
9128135129D2BE652809F5A1D337211AFFAD91ED5827474BF9BD7E285ECEF321
found!
```

La clef finale obtenue est donc 5E D4 9B 71 56 FC E4 7D E9 76 DA C5.
On voit bien l'entête en clair, et les 0xFF de la map du codage de huffman :

```
$ hexdump -C decrypted | head -5
00000000  42 5a 68 39 31 41 59 26  53 59 cc bd 29 cb 00 c2  |BZh91AY&SY..)...|
00000010  25 7f 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 6f 5f 00 0a  3e 8f 7d 8c fb 3d d6 1d  |....o_..>.}..=..|
00000040  b6 be 69 bb b7 9e ac b7  6b d6 75 7d bd ed 7d f6  |..i.....k.u}..}.|
```

Comme attendu, le fichier déchiffré est une archive BZip2 qui constituera notre stage 6.

```
$ file decrypted
decrypted: bzip2 compressed data, block size = 900k
$ cp decrypted ../stage6.tar.bz2
$ cd ..
```

2.5.2 Avec Miasm2

Il était également possible d'utiliser l'outil Miasm2 pour exécuter symboliquement les fonctions des transputers et en extraire les fonctions de transfert.

La première étape consiste à implémenter l'architecture ST20C4 (au moins une version amoindrie). Ceci implique d'écrire le code pour gérer les registres (`regs`), les instructions (`arch`), la sémentique (`sem`), les classes d'analyse de représentation intermédiaire (`ira`), le module de désassembleur (`disasm`), et le module d'exécution *Just-In-Time*[29] (`jit`).

L'implémentation des instructions `IN` et `OUT` peut être réalisée à travers le mécanisme de gestion d'exceptions CPU. Voici une souche de jitter ST20C4, qui servira ici de preuve de concept :

Listing 14 – src/stage5/m2/st20c4_jitter.py

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

from sys import stderr
from miasm2.jitter.csts       import *
from miasm2.analysis.machine import Machine
from miasm2.arch.st20c4.sem   import ST20C4_EXCEPTIONFLAG_IN, ST20C4_EXCEPTIONFLAG_OUT
from miasm2.core.bin_stream   import bin_stream_str

def do_in(curjit):
    regs = curjit.cpu.get_gpreg()
    print >> stderr, '\033[34min (%08X, %08X, %08X)\033[m' % (
        regs['Areg'], regs['Breg'], regs['Creg'])
    curjit.cpu.set_exception(0)
    return True
def do_out(curjit):
    regs = curjit.cpu.get_gpreg()
    print >> stderr, '\033[34mout(%08X, %08X, %08X)\033[m' % (
        regs['Areg'], regs['Breg'], regs['Creg'])
    curjit.cpu.set_exception(0)
    return True

debug   = lambda l: True   # l.name in ['IN', 'OUT']
machine = Machine('st20c4')
entrypt = 0x7FF80000
jitter  = machine.jitter()
jitter.jit.log_regs     = debug
jitter.jit.log_mn       = debug
jitter.jit.log_newbloc = debug
jitter.init_stack()
jitter.vm.add_memory_page(0x7FF80000, PAGE_READ | PAGE_WRITE, open('input.bin').read())
jitter.add_exception_handler(ST20C4_EXCEPTIONFLAG_IN,  do_in)
jitter.add_exception_handler(ST20C4_EXCEPTIONFLAG_OUT, do_out)
jitter.init_run(entrypt)

# initialisation values
jitter.cpu.Iptr  = entrypt
jitter.cpu.Wptr -= 4
jitter.cpu.Areg  = 0xAAAAAAAA
jitter.cpu.Breg  = 0xAAAAAAAA
jitter.cpu.Creg  = 0xAAAAAAAA

jitter.continue_run()
```

Le problème ici est que nous devrions lier entre eux 13 exemplaires concurrents de ce jitter. Voici notre implémentation de l'émulateur de réseau de transputers :

Listing 15 – src/stage5/m2/jserver.py

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

# reset ; ./jserver.py 2>&1 | grep 'HOST:RECV'

```

29. `http://en.wikipedia.org/wiki/Just-in-time_compilation`

```
6  from sys              import stderr
7  from re               import sub
8  from multiprocessing import Process, Pipe, Lock
9  from memorypool       import MemoryPool
10 from miasm2.arch.st20c4.arch import mn_st20c4
11
12 TESTVECTOR_KEY  = '5E541B71567C647D69760000'.decode('hex')
13 TESTVECTOR_DATA = open('encrypted', 'rb').read()
14
15 TESTVECTOR_KEY  = '*SSTIC-2015*'
16 TESTVECTOR_DATA = '1d87c4c4e0ee40383c59447f23798d9fefe74fb82480766e'.decode('hex')
17
18 class Host:
19     pipe = None
20     def run(self):
21         self.p = Process(target=self.mainloop)
22         self.p.start()
23     def join(self):
24         self.p.join()
25     def _recv(self):
26         data = self.pipe.recv()
27         lock.acquire()
28         print >> stderr, 'HOST:', 'RECV', '%r' % data
29         lock.release()
30         return data
31     def _send(self, data):
32         self.pipe.send(data)
33         lock.acquire()
34         print >> stderr, 'HOST:', 'SEND', '%r' % data
35         lock.release()
36     def mainloop(self):
37         self._recv()                    # 'Boot ok\x00'
38         self._send('\x00' * 12)         # end of code
39         self._recv()                    # 'Code ok\x00'
40         self._send('\x00' *  4)         # address ?
41         self._send(TESTVECTOR_KEY)      # key
42         self._recv()                    # 'Decrypt\x00'
43         self._send('\x0C')              # unknown
44         self._send('\x00' * 0xC)        # unknown
45         for c in TESTVECTOR_DATA:
46             self._send(c)
47             self._recv()
48         self._send('***END***' * 10)
49
50 class Transputer:
51     def __init__(self, tid, entry_point, wptr_offset = 4 * 6):
52         self.tid   = tid
53         self.links = [None] * 4
54         self.Iptr  = entry_point
55         self.Wptr  = 0x1FFFFFFC - wptr_offset
56         self.Areg  = 0xAAAAAAAA
57         self.Breg  = 0xAAAAAAAA
58         self.Creg  = 0xAAAAAAAA
59     def debug(self, op):
60         lock.acquire()
61         txtop = sub(r'  *', ' ', str(op)).ljust(20)
62         if   '*' in txtop:
63              txtop = '\033[33m%s\033[m' % txtop
64         elif txtop.strip() in ['IN', 'OUT']:
65              txtop = '\033[34m%s\033[m' % txtop
66         print >> stderr, 'TRANSPUTER%s:' % self.tid,
67         print >> stderr, '%s' % txtop,
68         print >> stderr, '(Areg=%08X, Breg=%08X, Creg=%08X, Wptr=%08X, Iptr=%08X)' % (
69             self.Areg, self.Breg, self.Creg, self.Wptr, self.Iptr)
70         lock.release()
71     def dumpmem(self, full=False):
72         lock.acquire()
73         if  full:
74             txt = memorypool
75         else:
```

```
76              txt = memorypool.pages[0x1FFFF000]
77              txt = '\n'.join('TRANSPUTER%s : %s' % (self.tid, line)
78                    for line in str(txt).splitlines()[-21:-1])
79          print >> stderr, txt
80          lock.release()
81      def run(self):
82          self.p = Process(target=self.mainloop)
83          self.p.start()
84      def join(self):
85          self.p.join()
86      def mainloop(self):
87          self.debug('***INIT***')
88          memorypool.write_uint32(self.Wptr, 0xAAAAAAAA)
89          while True:
90              asminstr = mn_st20c4.dis(memorypool.read(self.Iptr, 0x10), None)
91              if    False:
92                  pass
93              elif asminstr.name == 'ADC':
94                  self.Areg = (self.Areg + asminstr.args[0].arg.arg) & 0xFFFFFFFF
95              elif asminstr.name == 'AJW':
96                  self.Wptr = (self.Wptr + 4 * asminstr.args[0].arg.arg) & 0xFFFFFFFF
97              elif asminstr.name == 'AND':
98                  self.Areg, self.Breg = (self.Areg & self.Breg) & 0xFFFFFFFF, self.Creg
99              elif asminstr.name == 'BSUB':
100                 self.Areg, self.Breg = (self.Areg + self.Breg) & 0xFFFFFFFF, self.Creg
101             elif asminstr.name == 'CALL':
102                 self.Wptr -= 4 * 4
103                 memorypool.write_uint32(self.Wptr + 4 * 0, self.Iptr + 2)
104                 memorypool.write_uint32(self.Wptr + 4 * 1, self.Areg)
105                 memorypool.write_uint32(self.Wptr + 4 * 2, self.Breg)
106                 memorypool.write_uint32(self.Wptr + 4 * 3, self.Creg)
107                 self.Areg  =  self.Iptr + 2
108                 self.Iptr  = (self.Iptr + 0 + asminstr.args[0].arg.arg) & 0xFFFFFFFF
109             elif asminstr.name == 'CJ':
110                 if   self.Areg == 0:
111                     self.Iptr = (self.Iptr + asminstr.args[0].arg.arg) & 0xFFFFFFFF
112                 else:
113                     self.Areg, self.Breg = self.Breg, self.Creg
114             elif asminstr.name == 'DUP':
115                 self.Breg, self.Creg = self.Areg, self.Breg
116             elif asminstr.name == 'EQC':
117                 if   self.Areg == asminstr.args[0].arg.arg:
118                     self.Areg = 1
119                 else:
120                     self.Areg = 0
121             elif asminstr.name == 'GAJW':
122                 try:
123                     lock.acquire()
124                     old_areg, self.Areg = self.Areg, memorypool.read_uint32(self.Wptr)
125                     memorypool.write_uint32(self.Wptr, old_areg)
126                 except:
127                     print >> stderr, '\033[31mMEMORY ERROR IN TRANSPUTER %s\033[m' % self.tid
128                 finally:
129                     lock.release()
130             elif asminstr.name == 'GT':
131                 if   self.Breg > self.Areg:
132                     self.Areg = 1
133                 else:
134                     self.Areg = 0
135                 self.Breg = self.Creg
136             elif asminstr.name == 'J':
137                 self.Iptr = (self.Iptr + asminstr.args[0].arg.arg) & 0xFFFFFFFF
138             elif asminstr.name == 'LB':
139                 self.Areg = memorypool.read_uint8(self.Areg)
140             elif asminstr.name == 'LDC':
141                 self.Areg, self.Breg, self.Creg = \
142                 asminstr.args[0].arg.arg, self.Areg, self.Breg
143             elif asminstr.name == 'LDL':
144                 try:
145                     lock.acquire()
```

```
146                     addr = (self.Wptr + 4 * asminstr.args[0].arg.arg) & 0xFFFFFFFF
147                     self.Areg, self.Breg, self.Creg = \
148                     memorypool.read_uint32(addr), self.Areg, self.Breg
149                 except:
150                     self.Areg, self.Breg, self.Creg = 0xAAAAAAAA, self.Areg, self.Breg
151                     print >> stderr, '\033[31mMEMORY ERROR IN TRANSPUTER %s\033[m' % self.tid,
152                     print >> stderr, '(%s)' % hex(addr)
153                 finally:
154                     lock.release()
155             elif asminstr.name == 'LDLP':
156                 self.Areg, self.Breg, self.Creg = \
157                 (self.Wptr + 4 * asminstr.args[0].arg.arg) & 0xFFFFFFFF, self.Areg, self.Breg
158             elif asminstr.name == 'LDNLP':
159                 self.Areg = (self.Areg + 4 * asminstr.args[0].arg.arg) & 0xFFFFFFFF
160             elif asminstr.name == 'LDPI':
161                 self.Areg = (self.Areg + self.Iptr + 2) & 0xFFFFFFFF
162             elif asminstr.name == 'MINT':
163                 self.Areg, self.Breg, self.Creg = 0x80000000, self.Areg, self.Breg
164             elif asminstr.name == 'PROD':
165                 self.Areg, self.Breg = (self.Areg * self.Breg) & 0xFFFFFFFF, self.Creg
166             elif asminstr.name == 'REM':
167                 self.Areg, self.Breg = self.Breg % self.Areg, self.Creg # bug dans la doc
168             elif asminstr.name == 'RET':
169                 self.Iptr, self.Wptr = \
170                 (memorypool.read_uint32(self.Wptr) - 2) & 0xFFFFFFFF, self.Wptr + 4 * 4
171             elif asminstr.name == 'SB':
172                 memorypool.write_uint8(self.Areg, self.Breg & 0xFF)
173                 self.Areg = self.Creg
174             elif asminstr.name == 'SHR':
175                 self.Areg, self.Breg = self.Breg >> self.Areg, self.Creg
176             elif asminstr.name == 'SHL':
177                 self.Areg, self.Breg = self.Breg << self.Areg, self.Creg
178             elif asminstr.name == 'SSUB':
179                 self.Areg, self.Breg = (self.Areg + 2 * self.Breg) & 0xFFFFFFFF, self.Creg
180             elif asminstr.name == 'STL':
181                 addr = (self.Wptr + 4 * asminstr.args[0].arg.arg) & 0xFFFFFFFF
182                 memorypool.write_uint32(addr, self.Areg)
183                 self.Areg, self.Breg = self.Breg, self.Creg
184             elif asminstr.name == 'WSUB':
185                 self.Areg, self.Breg = (self.Areg + 4 * self.Breg) & 0xFFFFFFFF, self.Creg
186             elif asminstr.name == 'XOR':
187                 self.Areg, self.Breg = (self.Areg ^ self.Breg) & 0xFFFFFFFF, self.Creg
188             elif asminstr.name == 'IN':
189                 self.debug(asminstr)
190                 length, channel, offset = self.Areg, self.Breg, self.Creg
191                 link = (channel - 0x80000010) / 4
192                 data = self.links[link].recv()
193                 if  len(data) > length:
194                     lock.acquire()
195                     print >> stderr, '***END***'
196                     lock.release()
197                     self.dumpmem(True)
198                     exit(0)
199                 lock.acquire()
200                 print >> stderr, 'TRANSPUTER%s:' % self.tid, '\t[IN] ',
201                 print >> stderr, 'from link %d' % link, '%r' % data, data.encode('hex')
202                 lock.release()
203                 memorypool.write(offset, data)
204             elif asminstr.name == 'OUT':
205                 self.debug(asminstr)
206                 length, channel, offset = self.Areg, self.Breg, self.Creg
207                 link = (channel - 0x80000000) / 4
208                 data = memorypool.read(offset, length)
209                 lock.acquire()
210                 print >> stderr, 'TRANSPUTER%s:' % self.tid, '\t[OUT]',
211                 print >> stderr, 'to   link %d' % link, '%r' % data, data.encode('hex')
212                 lock.release()
213                 self.links[link].send(data)
214             else:
215                 self.debug('***ERROR***')
```

```
                    lock.acquire()
                    print >> stderr, 'TRANSPUTER%s :' % self.tid,
                    print >> stderr, '\033[31m%r\033[m' % memorypool.read(self.Iptr, 0x10)
                    lock.release()
                    break
                self.Iptr += asminstr.l
                self.debug(asminstr)
                if   asminstr.name in ['CALL','GAJW','SB','STL','IN',]:
                    self.dumpmem()


if __name__ == '__main__':

    memorypool = MemoryPool()
    memorypool.write(0x7FF80000, open('input.bin', 'rb').read())

    lock = Lock()

    Transputers     = [None] * 16
    Localhost       = Host()

    Transputers[0]  = Transputer(0,  0x7FF80000)
    Transputers[1]  = Transputer(1,  0x7FF80105)
    Transputers[2]  = Transputer(2,  0x7FF80182)
    Transputers[3]  = Transputer(3,  0x7FF801FF)
    Transputers[4]  = Transputer(4,  0x7FF804C5)
    Transputers[5]  = Transputer(5,  0x7FF8052D)
    Transputers[6]  = Transputer(6,  0x7FF80595)
    Transputers[7]  = Transputer(7,  0x7FF80639)
    Transputers[8]  = Transputer(8,  0x7FF806B5)
    Transputers[9]  = Transputer(9,  0x7FF80769)
    Transputers[10] = Transputer(10, 0x7FF807D5)
    Transputers[11] = Transputer(11, 0x7FF80885)
    Transputers[12] = Transputer(12, 0x7FF8090D)

    Localhost.pipe,           Transputers[0].links[0]  = Pipe()
    Transputers[0].links[1],  Transputers[1].links[0]  = Pipe()
    Transputers[0].links[2],  Transputers[2].links[0]  = Pipe()
    Transputers[0].links[3],  Transputers[3].links[0]  = Pipe()
    Transputers[1].links[1],  Transputers[4].links[0]  = Pipe()
    Transputers[1].links[2],  Transputers[5].links[0]  = Pipe()
    Transputers[1].links[3],  Transputers[6].links[0]  = Pipe()
    Transputers[2].links[1],  Transputers[7].links[0]  = Pipe()
    Transputers[2].links[2],  Transputers[8].links[0]  = Pipe()
    Transputers[2].links[3],  Transputers[9].links[0]  = Pipe()
    Transputers[3].links[1],  Transputers[10].links[0] = Pipe()
    Transputers[3].links[2],  Transputers[11].links[0] = Pipe()
    Transputers[3].links[3],  Transputers[12].links[0] = Pipe()
    Transputers[11].links[1], Transputers[12].links[1] = Pipe()

    Localhost.run()
    for t in Transputers:
        if  t is not None:
            t.run()
```

La communication entre les transputers est assurée par le système de pipes fourni par le module standard de multi-processing en Python. L'émulation est effectuée à la main, avec des variables classiques du langage et une implémentation de pool mémoire. Si on l'exécute (c'est très très lent) :

```
$ ./jserver.py 2>&1 | grep 'HOST:RECV'
HOST:RECV 'Boot ok\x00'
HOST:RECV 'Code Ok\x00'
HOST:RECV 'Decrypt\x00'
HOST:RECV 'I'
HOST:RECV ' '
HOST:RECV 'l'
HOST:RECV 'o'
HOST:RECV 'v'
HOST:RECV 'e'
HOST:RECV ' '
```

```
HOST : RECV 'S'
HOST : RECV 'T'
HOST : RECV '2'
HOST : RECV '0'
HOST : RECV ' '
HOST : RECV 'a'
HOST : RECV 'r'
HOST : RECV 'c'
HOST : RECV 'h'
HOST : RECV 'i'
HOST : RECV 't'
HOST : RECV 'e'
HOST : RECV 'c'
HOST : RECV 't'
HOST : RECV 'u'
HOST : RECV 'r'
HOST : RECV 'e'
^C
```

Il était aussi possible de générer les fonctions de tranfert de chaque transputer.
Voici le script qui sert à générer le graphe :

Listing 16 – src/stage5/m2/run.sh

```
#!/bin/bash

(
echo "digraph sstic2015 {"
echo "size=\"80,50\";"
echo "fontsize = \"16\";"
echo "node ["
echo "shape = \"box\""
echo "];"

for FUNC in \
    SUB_000 \
    SUB_105 SUB_182 SUB_1FF \
    SUB_4C5 SUB_52D SUB_595 SUB_639 SUB_6B5 SUB_769 SUB_7D5 SUB_885 SUB_90D
do
    echo $FUNC "["
    echo "label=\"$FUNC\\l\\"
    ./dirtyemulbloc.py $FUNC | sed 's#$#\\l\\#g'
    echo "\""
    echo "];"
done

echo "HOST    -> SUB_000 [ taillabel=\"\",  headlabel=\"0\" ];"
echo "SUB_000 -> SUB_105 [ taillabel=\"1\", headlabel=\"0\" ];"
echo "SUB_000 -> SUB_182 [ taillabel=\"2\", headlabel=\"0\" ];"
echo "SUB_000 -> SUB_1FF [ taillabel=\"3\", headlabel=\"0\" ];"
echo "SUB_105 -> SUB_4C5 [ taillabel=\"1\", headlabel=\"0\" ];"
echo "SUB_105 -> SUB_52D [ taillabel=\"2\", headlabel=\"0\" ];"
echo "SUB_105 -> SUB_595 [ taillabel=\"3\", headlabel=\"0\" ];"
echo "SUB_182 -> SUB_639 [ taillabel=\"1\", headlabel=\"0\" ];"
echo "SUB_182 -> SUB_6B5 [ taillabel=\"2\", headlabel=\"0\" ];"
echo "SUB_182 -> SUB_769 [ taillabel=\"3\", headlabel=\"0\" ];"
echo "SUB_1FF -> SUB_7D5 [ taillabel=\"1\", headlabel=\"0\" ];"
echo "SUB_1FF -> SUB_885 [ taillabel=\"2\", headlabel=\"0\" ];"
echo "SUB_1FF -> SUB_90D [ taillabel=\"3\", headlabel=\"0\" ];"
echo "SUB_885 -> SUB_90D [ taillabel=\"1\", headlabel=\"1\" ];"

echo "}"
) > whole_arch.gv
```

Et voici le script qui détermine la fonction de transfert d'une fonction. L'implémentation est basée sur une pile simplifiée. Les trois registres Areg, Breg et Creg forment une pile que l'on émule par une simple liste Python.

Listing 17 – src/stage5/m2/dirtyemulbloc.py

```
#!/usr/bin/env python
#-*- coding :utf-8 -*-

from sys import stderr, argv
from miasm2.arch.st20c4.regs import *
from miasm2.arch.st20c4.arch import *
from miasm2.analysis.machine import Machine
from miasm2.jitter.csts import *
from miasm2.arch.st20c4.disasm import dis_st20c4
from miasm2.arch.st20c4.sem import ST20C4_EXCEPTIONFLAG_IN, ST20C4_EXCEPTIONFLAG_OUT
from miasm2.core.bin_stream import bin_stream_str
from miasm2.expression.expression import ExprInt32, ExprCompose
from miasm2.expression.simplifications import expr_simp


SUB_000 = [0x00,0x23,0x38,0xD3]     # Transputer 0

SUB_105 = [0x105,0x11A,0x12C]       # Transputer 1
SUB_182 = [0x182,0x197,0x1A9]       # Transputer 2
SUB_1FF = [0x1FF,0x214,0x226]       # Transputer 3

SUB_4C5 = [0x4B9,0x4BF,0x4C5,0x4ED,0x4EF]
SUB_52D = [0x521,0x527,0x52D,0x555,0x557]
SUB_595 = [0x589,0x58F,0x595,0x5AA,0x5C0,0x5C3]
SUB_639 = [0x62D,0x633,0x639,0x66A,0x66C]
SUB_6B5 = [0x6A9,0x6AF,0x6B5,0x6D0,0x6D2,0x6D9,0x6DB,0x6F0,0x713,0x716,0x728,0x72B]
SUB_769 = [0x75D,0x763,0x769,0x797,0x799]
SUB_7D5 = [0x7C9,0x7CF,0x7D5,0x7F0,0x7F2,0x7F9,0x7FB,0x810,0x82B,0x82E]
SUB_885 = [0x879,0x87F,0x885]
SUB_90D = [0x901,0x907,0x90D,0x922,0x924]

SUBS    = [eval(argv[1]),]

filedata  = open('input.bin', 'rb').read()
stream    = bin_stream_str(filedata)
disengine = dis_st20c4(stream)
for sub in SUBS:
    for ad in sub:
        disasm = disengine.dis_bloc(ad)
        print disasm.label
        stack, pc, sp = [Areg_init, Breg_init, Creg_init], \
            Iptr_init + ExprInt32(disasm.label.offset), Wptr_init
        for line in disasm.lines:
            #print '\t', '%08X' % line.offset, line
            if   False:
                pass
            elif line.name == 'ADC':
                stack[0] += line.args[0]
            elif line.name == 'AJW':
                sp += ExprInt32(4) * line.args[0]
            elif line.name == 'AND':
                stack = [stack[0] & stack[1], stack[2], stack[2]]
            elif line.name == 'BSUB':
                stack = [stack[0] + stack[1], stack[2], stack[2]]
            elif line.name == 'CALL':
                print '\t', 'call to %s' % expr_simp((pc + ExprInt32(line.l) + line.args[0]))
            elif line.name == 'CJ':
                print '\t', 'jmp to %s == 0 ? %s  : %s' % (stack[0], line.args[0], 'next')
            elif line.name == 'DUP':
                stack = [stack[0], stack[0], stack[1]]
            elif line.name == 'EQC':
                print '\t', 'check if %s == %s' % (stack[0], line.args[0])
            elif line.name == 'GAJW':
                tmp, stack[0] = stack[0], ExprMem(Wptr)
                print '\t', '@[%s] = %s' % (expr_simp(sp), tmp)
            elif line.name == 'GCALL':
                pc, stack[0] = stack[0], pc
                print '\t', 'jmp (gcall) to %s' % (pc)
            elif line.name == 'GT':
```

```
                print '\t', 'check if %s > %s' % (stack[1], stack[0])
            elif line.name == 'IN':
                print '\t', 'in (%s)' % map(str, stack)
            elif line.name == 'J':
                pc = line.args[0]
                print '\t', 'jmp (j) to %s' % (pc)
            elif line.name == 'LB':
                stack[0] = ExprCompose([(ExprInt_fromsize(24, 0), 0, 24),
                                        (ExprMem(stack[0], 8)[:8], 24, 32)])
            elif line.name == 'LDC':
                stack = [line.args[0]] + stack[:2]
            elif line.name == 'LDL':
                stack = [ExprMem(sp + ExprInt32(4) * line.args[0], 32)] + stack[:2]
            elif line.name == 'LDLP':
                stack = [sp + ExprInt32(4) * line.args[0]] + stack[:2]
            elif line.name == 'LDNLP':
                stack[0] += ExprInt32(4) * line.args[0]
            elif line.name == 'LDPI':
                stack[0] += pc + ExprInt32(2)
            elif line.name == 'MINT':
                stack = [ExprInt32(0x80000000)] + stack[:2]
            elif line.name == 'OUT':
                if   str(stack[2]) == '(Iptr_init+0xDD)':
                    print '\t', 'out(%s)' % map(str, stack[:2] + ['Boot ok\\x00'])
                elif str(stack[2]) == '(Iptr_init+0xE5)':
                    print '\t', 'out(%s)' % map(str, stack[:2] + ['Code ok\\x00'])
                elif str(stack[2]) == '(Iptr_init+0xED)':
                    print '\t', 'out(%s)' % map(str, stack[:2] + ['Decrypt\\x00'])
                else:
                    print '\t', 'out(%s)' % map(str, stack)
            elif line.name == 'PROD':
                stack[0], stack[1] = stack[0] * stack[1], stack[2]
            elif line.name == 'SB':
                tmp, stack[0] = stack[0], stack[2]
                print '\t', '@[%s] = %s' % (expr_simp(ExprMem(tmp, 8)), stack[1][:8])
            elif line.name == 'SSUB':
                stack = [stack[0] + ExprInt32(2) * stack[1], stack[2], stack[2]]
            elif line.name == 'STL':
                tmp, stack = stack[0], stack[-2:] + [stack[2]]
                print '\t', '@[%s] = %s' % (expr_simp(sp + ExprInt32(4) * line.args[0]), tmp)
            elif line.name == 'XOR':
                stack = [stack[0] ^ stack[1], stack[2], stack[2]]
            else:
                print '\t', '%08X' % line.offset, line
            stack[0] = expr_simp(stack[0])
            stack[1] = expr_simp(stack[1])
            stack[2] = expr_simp(stack[2])
            pc += ExprInt32(line.l)
            pc  = expr_simp(pc)
            sp  = expr_simp(sp)
            #print '\t', map(str, stack), pc, sp
        #for bto in disasm.bto:
        #    print bto
        print '=' * 70
    #for _ in xrange(3):
    #    print
```

Avec la retranscription symbolique de Miasm2, voilà à quoi ressemblent quelques blocs :

```
loc_0000000000000000 :0x00000000
        @[(Wptr_init+0xFFFFFED4)] = 0x0
        @[(Wptr_init+0xFFFFFEDC)] = 0x0
        @[(Wptr_init+0xFFFFFED0)] = 0x80001000
        out(['0x8', '0x80000000', 'Boot ok\\x00'])
        in (['0xC', '0x80000010', '(Wptr_init+0xFFFFFEC4)'])
        jmp to @32[(Wptr_init+0xFFFFFEC4)] == 0 ? loc_0000000000000038 :0x00000038 \
: next
================================================================================
loc_0000000000000023 :0x00000023
        in (['@32[(Wptr_init+0x124)]', '0x80000010', '(Iptr_init+0xF4)'])
```

```
        out(['@32[(Wptr_init+0x124)]', '@32[(Wptr_init+0x128)]', '(Iptr_init+0xF4)\
'])
        jmp (j) to loc_0000000000000018 :0x00000018
=============================================================================
loc_0000000000000038 :0x00000038
        out(['0xC', '0x80000004', '(Wptr_init+0x124)'])
        out(['0xC', '0x80000008', '(Wptr_init+0x124)'])
        out(['0xC', '0x8000000C', '(Wptr_init+0x124)'])
        out(['0x8', '0x80000000', 'Code ok\\x00'])
        in (['0x4', '0x80000010', '(Wptr_init+0x8)'])
        in (['0xC', '0x80000010', '(Wptr_init+0x14)'])
        out(['0x8', '0x80000000', 'Decrypt\\x00'])
        in (['0x1', '0x80000010', '(Wptr_init+0xC)'])
        in (['{0x0,0,24, @8[(Wptr_init+0xC)],24,32}', '0x80000010', '(Wptr_init+0x\
24)'])
        @[(Wptr_init+0x10)] = 0x0
        in (['0x1', '0x80000010', '(Wptr_init+0x4)'])
        out(['0xC', '0x80000004', '(Wptr_init+0x14)'])
        out(['0xC', '0x80000008', '(Wptr_init+0x14)'])
        out(['0xC', '0x8000000C', '(Wptr_init+0x14)'])
        in (['0x1', '0x80000014', '(Wptr_init+0x1)'])
        in (['0x1', '0x80000018', '(Wptr_init+0x2)'])
        in (['0x1', '0x8000001C', '(Wptr_init+0x3)'])
        @[@8[(Wptr_init+0x1)]] = {0x0,0,24, (@8[(Wptr_init+0x1)]^@8[(Wptr_init+0x2\
)]^@8[(Wptr_init+0x3)]),24,32}[0:8]
        @[@8[Wptr_init]] = ((@32[(Wptr_init+0x10)]+({0x0,0,24, @8[(Wptr_init+@32[(\
Wptr_init+0x10)]+0x14)],24,32}*0x2))^{0x0,0,24, @8[(Wptr_init+0x4)],24,32})\
[0:8]
        @[@8[(Wptr_init+@32[(Wptr_init+0x10)]+0x14)]] = {0x0,0,24, @8[(Wptr_init+0\
x1)],24,32}[0:8]
        @[(Wptr_init+0x10)] = (@32[(Wptr_init+0x10)]+0x1)
        check if (@32[(Wptr_init+0x10)]+0x1) == 0xC
        jmp to (@32[(Wptr_init+0x10)]+0x1) == 0 ? loc_00000000000000D6 :0x000000d6 \
: next
=============================================================================
loc_00000000000000D3 :0x000000d3
        @[(Wptr_init+0x10)] = 0x0
        out(['0x1', '0x80000000', 'Wptr_init'])
        jmp (j) to loc_0000000000000078 :0x00000078
=============================================================================


loc_0000000000000105 :0x00000105
        @[(Wptr_init+0xFFFFFFE0)] = 0x80001000
        in (['0xC', '0x80000010', '(Wptr_init+0xFFFFFFD4)'])
        jmp to @32[(Wptr_init+0xFFFFFFD4)] == 0 ? loc_000000000000012C :0x0000012c \
: next
=============================================================================
loc_000000000000011A :0x0000011a
        in (['@32[(Wptr_init+0x14)]', '0x80000010', '(Iptr_init+0x172)'])
        out(['@32[(Wptr_init+0x14)]', '@32[(Wptr_init+0x18)]', '(Iptr_init+0x172)'\
])
        jmp (j) to loc_0000000000000111 :0x00000111
=============================================================================
loc_000000000000012C :0x0000012c
        in (['0xC', '0x80000010', '(Wptr_init+0x4)'])
        out(['0xC', '0x80000004', '(Wptr_init+0x4)'])
        out(['0xC', '0x80000008', '(Wptr_init+0x4)'])
        out(['0xC', '0x8000000C', '(Wptr_init+0x4)'])
        in (['0x1', '0x80000014', '(Wptr_init+0x1)'])
        in (['0x1', '0x80000018', '(Wptr_init+0x2)'])
        in (['0x1', '0x8000001C', '(Wptr_init+0x3)'])
        @[@8[Wptr_init]] = {0x0,0,24, (@8[(Wptr_init+0x1)]^@8[(Wptr_init+0x2)]^@8[\
(Wptr_init+0x3)]),24,32}[0:8]
        out(['0x1', '0x80000000', 'Wptr_init'])
        jmp (j) to loc_000000000000012C :0x0000012c
=============================================================================


loc_00000000000004B9 :0x000004b9
```

```
        in (['@32[(Wptr_init+0x10)]', '@32[(Wptr_init+0x8)]', '@32[(Wptr_init+0xC)\
]'])
        000004BD RET
================================================================================
loc_00000000000004BF :0x000004bf
        out(['@32[(Wptr_init+0x10)]', '@32[(Wptr_init+0x8)]', '@32[(Wptr_init+0xC)\
]'])
        000004C3 RET
================================================================================
loc_00000000000004C5 :0x000004c5
        @[(Wptr_init+0xFFFFFFF0)] = 0x0
        @[@8[(Wptr_init+0xFFFFFFF0)]] = 0x0[0:8]
        @[(Wptr_init+0xFFFFFFEC)] = 0xC
        call to (Iptr_init+0x4B9)
        @[(Wptr_init+0xFFFFFFEC)] = 0x0
        @[@8[(Wptr_init+0xFFFFFFF0)]] = (({0x0,0,24, @8[(Wptr_init+@32[(Wptr_init+\
0xFFFFFFEC)]+0xFFFFFFF4)],24,32}+{0x0,0,24, @8[(Wptr_init+0xFFFFFFF0)],24,3\
2})&0xFF)[0:8]
        @[(Wptr_init+0xFFFFFFEC)] = (@32[(Wptr_init+0xFFFFFFEC)]+0x1)
        check if 0xC > @32[(Wptr_init+0xFFFFFFEC)]
        jmp to @32[(Wptr_init+0xFFFFFFEC)] == 0 ? loc_00000000000004EF :0x000004ef \
: next
================================================================================
loc_00000000000004ED :0x000004ed
        jmp (j) to loc_00000000000004D8 :0x000004d8
================================================================================
loc_00000000000004EF :0x000004ef
        @[Wptr_init] = 0x1
        call to (Iptr_init+0x4BF)
        jmp (j) to loc_00000000000004CD :0x000004cd
================================================================================


loc_0000000000000521 :0x00000521
        in (['@32[(Wptr_init+0x10)]', '@32[(Wptr_init+0x8)]', '@32[(Wptr_init+0xC)\
]'])
        00000525 RET
================================================================================
loc_0000000000000527 :0x00000527
        out(['@32[(Wptr_init+0x10)]', '@32[(Wptr_init+0x8)]', '@32[(Wptr_init+0xC)\
]'])
        0000052B RET
================================================================================
loc_000000000000052D :0x0000052d
        @[(Wptr_init+0xFFFFFFF0)] = 0x0
        @[@8[(Wptr_init+0xFFFFFFF0)]] = 0x0[0:8]
        @[(Wptr_init+0xFFFFFFEC)] = 0xC
        call to (Iptr_init+0x521)
        @[(Wptr_init+0xFFFFFFEC)] = 0x0
        @[@8[(Wptr_init+0xFFFFFFF0)]] = ((@32[(Wptr_init+0xFFFFFFF0)]^{0x0,0,24, @\
8[(Wptr_init+@32[(Wptr_init+0xFFFFFFEC)]+0xFFFFFFF4)],24,32})&0xFF)[0:8]
        @[(Wptr_init+0xFFFFFFEC)] = (@32[(Wptr_init+0xFFFFFFEC)]+0x1)
        check if 0xC > @32[(Wptr_init+0xFFFFFFEC)]
        jmp to @32[(Wptr_init+0xFFFFFFEC)] == 0 ? loc_0000000000000557 :0x00000557 \
: next
================================================================================
loc_0000000000000555 :0x00000555
        jmp (j) to loc_0000000000000540 :0x00000540
================================================================================
loc_0000000000000557 :0x00000557
        @[Wptr_init] = 0x1
        call to (Iptr_init+0x527)
        jmp (j) to loc_0000000000000535 :0x00000535
================================================================================


loc_000000000000062D :0x0000062d
        in (['@32[(Wptr_init+0x10)]', '@32[(Wptr_init+0x8)]', '@32[(Wptr_init+0xC)\
]'])
        00000631 RET
```

```
================================================================================
loc_0000000000000633 :0x00000633
        out(['@32[(Wptr_init+0x10)]', '@32[(Wptr_init+0x8)]', '@32[(Wptr_init+0xC)\
]'])
        00000637 RET
================================================================================
loc_0000000000000639 :0x00000639
        @[(Wptr_init+0xFFFFFFF0)] = 0x0
        @[(Wptr_init+0xFFFFFFE4)] = 0xC
        call to (Iptr_init+0x62D)
        @[(Wptr_init+0xFFFFFFE8)] = 0x0
        @[(Wptr_init+0xFFFFFFEC)] = 0x0
        @[(Wptr_init+0xFFFFFFE4)] = 0x0
        @[(Wptr_init+0xFFFFFFE8)] = ((@32[(Wptr_init+0xFFFFFFE8)]+{0x0,0,24, @8[(W\
ptr_init+@32[(Wptr_init+0xFFFFFFE4)]+0xFFFFFFF4)],24,32})&0xFF)
        @[(Wptr_init+0xFFFFFFEC)] = ((@32[(Wptr_init+0xFFFFFFEC)]+{0x0,0,24, @8[(W\
ptr_init+@32[(Wptr_init+0xFFFFFFE4)]+0xFFFFFFFA)],24,32})&0xFF)
        @[(Wptr_init+0xFFFFFFE4)] = (@32[(Wptr_init+0xFFFFFFE4)]+0x1)
        check if 0x6 > @32[(Wptr_init+0xFFFFFFE4)]
        jmp to @32[(Wptr_init+0xFFFFFFE4)] == 0 ? loc_000000000000066C :0x0000066c \
: next
================================================================================
loc_000000000000066A :0x0000066a
        jmp (j) to loc_000000000000064C :0x0000064c
================================================================================
loc_000000000000066C :0x0000066c
        @[@8[(Wptr_init+0xC)]] = ((@32[(Wptr_init+0x4)]^@32[(Wptr_init+0x8)])&0xFF\
)[0:8]
        @[Wptr_init] = 0x1
        call to (Iptr_init+0x633)
        jmp (j) to loc_000000000000063D :0x0000063d
================================================================================
```

Et sous la forme d'un graphe global du réseau de transputers :

Toute la pull request pour Miasm2 est disponible en annexe.

2.5.3 Commentaire sur jserver

Une de mes premières tentatives était de tenter de faire fonctionner l'émulateur de transputer jserver [30]. On le configure en lui fournissant un fichier ipc.net :

Listing 18 – src/stage5/ipc.net

```
PROCESSOR 0
LINK 0 Host
LINK 1 Link01 Link10
LINK 2 Link02 Link20
LINK 3 Link03 Link30

PROCESSOR 1
LINK 0 Link10 Link01
LINK 1 Link14 Link41
LINK 2 Link15 Link51
LINK 3 Link16 Link61

PROCESSOR 2
LINK 0 Link20 Link02
LINK 1 Link27 Link72
LINK 2 Link28 Link82
LINK 3 Link29 Link92

PROCESSOR 3
LINK 0 Link30 Link03
LINK 1 Link310 Link103
LINK 2 Link311 Link113
LINK 3 Link312 Link123

PROCESSOR 4
LINK 0 Link41 Link14
LINK 1 -
LINK 2 -
LINK 3 -

PROCESSOR 5
LINK 0 Link51 Link15
LINK 1 -
LINK 2 -
LINK 3 -

PROCESSOR 6
LINK 0 Link61 Link16
LINK 1 -
LINK 2 -
LINK 3 -

PROCESSOR 7
LINK 0 Link72 Link27
LINK 1 -
LINK 2 -
LINK 3 -

PROCESSOR 8
LINK 0 Link82 Link28
LINK 1 -
LINK 2 -
LINK 3 -

PROCESSOR 9
LINK 0 Link92 Link29
LINK 1 -
LINK 2 -
LINK 3 -
```

30. https://sites.google.com/site/transputeremulator/Home/multiprocessor-jserver-support

```
60
61 PROCESSOR 10
62 LINK 0 Link103 Link310
63 LINK 1 -
64 LINK 2 -
65 LINK 3 -
66
67 PROCESSOR 11
68 LINK 0 Link113 Link311
69 LINK 1 Link1112 Link1211
70 LINK 2 -
71 LINK 3 -
72
73 PROCESSOR 12
74 LINK 0 Link123 Link312
75 LINK 1 Link1211 Link1112
76 LINK 2 -
77 LINK 3 -
78 #
```

On lance tous les transputers de la manière suivante :

Listing 19 – src/stage5/runjserver.bat

```
1  @start cmd /K jserver /sn 1 /si
2  @start cmd /K jserver /sn 2 /si
3  @start cmd /K jserver /sn 3 /si
4  @start cmd /K jserver /sn 4 /si
5  @start cmd /K jserver /sn 5 /si
6  @start cmd /K jserver /sn 6 /si
7  @start cmd /K jserver /sn 7 /si
8  @start cmd /K jserver /sn 8 /si
9  @start cmd /K jserver /sn 9 /si
10 @start cmd /K jserver /sn 10 /si
11 @start cmd /K jserver /sn 11 /si
12 @start cmd /K jserver /sn 12 /si
13 @timeout 3 /NOBREAK > nul
14 @jserver /sb sstic2015.btl /sn 0 /sz /sm /sw | more
```

Mais en vain, je n'ai pas réussi à lui faire émuler le bundle ST20.

2.5.4 Commentaire sur st20emu

L'implémentation de certaines fonctionnalités avec Miasm2 ont été débugguées avec st20emu [31], bien que cet outil n'implémente pas les instructions IN et OUT, et qu'il faille les simuler en plaçant des breakpoints dessus et en lisant ou écrivant la mémoire à l'instant où le breakpoint est rencontré. C'est un outil utile pour le debugging, mais peu pratique pour l'émulation en environnement réel.

2.6 Stage 6 - Stéganographie

Nous pouvons commencer par copier le stage 6 dans un répertoire dédié, vérifier son intégrité et l'extraire :

```
$ mkdir -p stage6
$ cp stage6.tar.bz2 stage6/
$ cd stage6
$ tar tvjf stage6.tar.bz2
-rw-r--r-- test/test    252569 2015-03-23 10:34 congratulations.jpg
$ tar xjf stage6.tar.bz2
$ rm -f stage6.tar.bz2
$ file *
congratulations.jpg: JPEG image data, JFIF standard 1.01
```

Un seul fichier, et il s'agit d'une image JPG. Regardons ce qu'elle contient.

31. http://sourceforge.net/projects/st20emu

FIGURE 5 – src/stage6/congratulations.jpg

Il est possible qu'il s'agisse de stéganographie[32]. J'ai donc téléchargé et installé *stegdetect*[33] (qui est particulièrement dédié au format JPG), et je l'ai lancé en mode détection sur le fichier.

```
$ ./stegdetect congratulations.jpg
congratulations.jpg  : appended(2096)<[nonrandom][data][BZh91AY&SY.....]>
```

Ça alors ! Un Autre BZip2 est concaténé à la fin des données utiles de l'image ! On l'extrait donc :

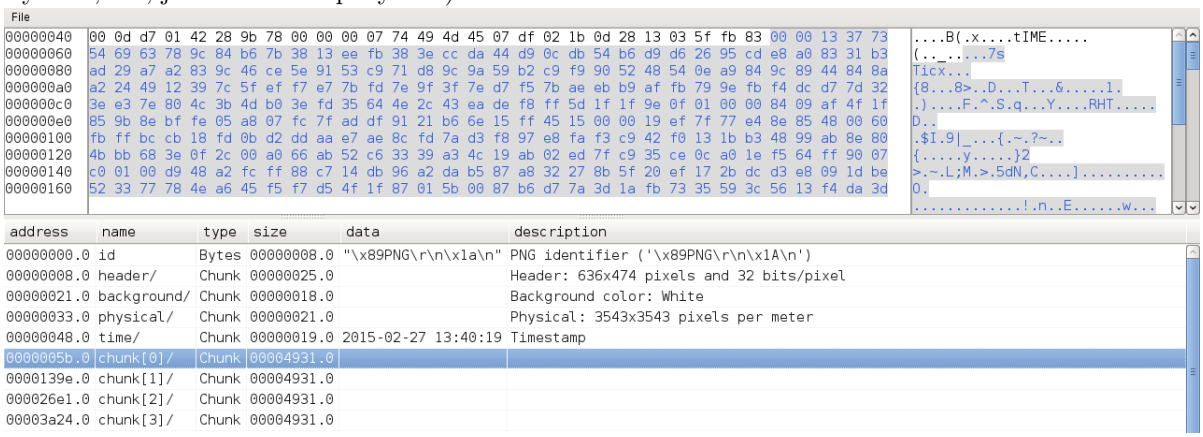
```
$ python -c "d=open('congratulations.jpg', 'rb').read() ;
  open('embedded1.tar.bz2', 'wb').write(d[d.index('BZh'):])"
$ tar tvjf embedded1.tar.bz2
-rw-r--r-- test/test     197557 2015-03-23 10:34 congratulations.png
$ tar xjf embedded1.tar.bz2
$ rm -f embedded1.tar.bz2
```

Une nouvelle image ! Regardons ce qu'elle contient :

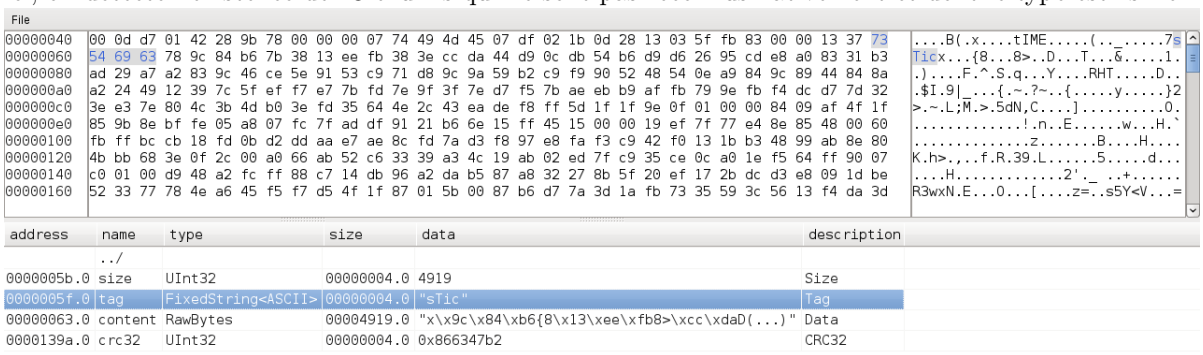
FIGURE 6 – src/stage6/congratulations.png

32. http://fr.wikipedia.org/wiki/Stéganographie#Message_transport.C3.A9_dans_une_image
33. http://www.outguess.org/download.php

Arf ! Les concepteurs ont de l'humour. Après quelques recherches infructueuses sur un contenu incorporé, je choisis d'ouvrir le fichier PNG dans hachoir [34] (qui est un dissecteur multi-format écrit en Python ; oui, j'aime beaucoup Python).

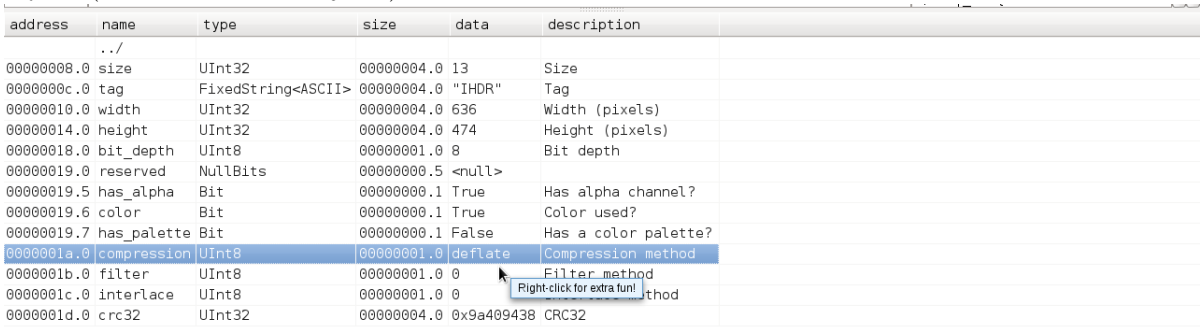


Un fichier PNG [35] est constitué de chunks de données, et chaque chunk est taggé avec un type préci sur 4 octets (IHDR pour l'entête du fichier, PLTE pour la palette, IEND pour la fin du fichier, etc.). Or ici, on détecte l'existence de 28 chunks qui ne sont pas reconnus nativement et dont le type est "sTic".



Chaque chunk est encodé comme suit :
- la taille du bloc sur 4 octets ;
- le type du bloc sur 4 octets ;
- les données ;
- le CRC32 du bloc sur 4 octets.

Un indice dans l'entête du fichier PNG stipule que les données sont compressées avec la méthode *deflate* (un standard de nos jours).



Il faut donc extraire et concaténer les données de tous les blocs sTic et décompresser le résultat. On implémente un petit script pour automatiser cette tâche :

34. https://bitbucket.org/haypo/hachoir/src
35. http://en.wikipedia.org/wiki/Portable_Network_Graphics

Listing 20 – src/stage6/pngextract.py

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

"Extract hidden data from PNG picture"

__version__ = '0.1'

import zlib

def main():
    "main function"
    with open('congratulations.png', 'rb') as _fd:
        data   = _fd.read()[0x5B:0x2089E]
        result = ''
        for i in xrange(0, len(data), 4931):
            chunk   = data[i:i+4931]
            result += chunk[8:-4]
        open('embedded', 'wb').write(zlib.decompress(result))

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

Il faut maintenant déterminer ce que l'on a extrait précisément :

```
$ file embedded
embedded: bzip2 compressed data, block size = 900k
$ tar tvjf embedded
-rw-r--r-- test/test     904520 2015-03-23 10:34 congratulations.tiff
$ tar xjf embedded
$ rm -f embedded
```

Il s'agit donc d'une troisième image, au format TIFF :

FIGURE 7 – src/stage6/congratulations.tiff

Arf! Encore. Bien. Le TIFF [36] est un format relativement complexe et très extensible. Essayons de voir dans gimp ce qu'on peut découvrir.

On remarque que si on utilise le pot de peinture (avec un seuil de 0) pour appliquer du rouge sur les zones unies, elles ne sont pas si unies que cela. Une technique habituellement utilisée en stéganographie

36. http://en.wikipedia.org/wiki/Tagged_Image_File_Format

consiste à encoder les bits du message cachés dans le bit de poids faible des octets de données de l'image.

FIGURE 8 – Avec l'outil « pot de peinture » de Gimp ou en augmentant à fond la luminosité

En affichant les triplets (R, G, B) des couleurs présentes dans l'image, on remarque quelque chose de particulier :

```
(0, 1, 0) (0, 0, 0) (0, 0, 0) (1, 0, 0) (0, 1, 0) (0, 1, 0) (1, 0, 0)
(1, 0, 0) (0, 1, 0) (1, 0, 0) (1, 0, 0) (0, 0, 0) (0, 0, 0) (1, 1, 0)
(1, 0, 0) (0, 1, 0) (0, 0, 0) (1, 1, 0) (0, 0, 0) (0, 1, 0) (0, 1, 0)
(0, 0, 0) (0, 0, 0) (0, 1, 0) (0, 1, 0) (0, 1, 0) (1, 0, 0) (0, 1, 0)
(0, 0, 0) (1, 0, 0) (0, 1, 0) (1, 0, 0) (0, 1, 0) (0, 1, 0) (0, 0, 0)
(1, 1, 0) (0, 1, 0) (0, 1, 0) (1, 0, 0) (0, 1, 0) (0, 1, 0) (0, 0, 0)
(0, 0, 0) (1, 0, 0) (0, 0, 0) (0, 0, 0) (0, 0, 0) (1, 0, 0) (0, 0, 0)
[...]
(1, 0, 0) (1, 1, 0) (0, 1, 0) (1, 0, 0) (1, 0, 0) (1, 1, 0) (1, 1, 0)
(0, 1, 0) (1, 1, 0) (0, 1, 0) (1, 1, 0) (1, 0, 0) (0, 1, 0) (1, 1, 0)
(1, 0, 0) (1, 1, 0) (1, 1, 0) (0, 1, 0) (1, 1, 0) (1, 0, 0) (1, 0, 0)
(1, 1, 0) (1, 1, 0) (1, 0, 0) (0, 1, 0) (0, 1, 0) (0, 1, 0) (0, 1, 0)
(0, 1, 0) (1, 0, 0) (1, 1, 0) (1, 0, 0) (1, 1, 0) (1, 1, 0) (0, 1, 0)
```

Il semble que le bit de poids faible de la composante bleue ne soit pas utilisé pour encoder les données cachées. Nous allons donc implémenter un script pour automatiser l'extraction des données :

Listing 21 – src/stage6/tiffextract.py

```
1  #!/usr/bin/env python
2  #-*- coding :utf-8 -*-
3
4  "Extract hidden data from TIFF picture"
5
6  __version__ = '0.1'
7
8  import PIL.Image
9
10 def main():
11     "main function"
12     binresult, result = '', ''
13     img = PIL.Image.open('congratulations.tiff')
14     width, height = img.size
15     for y in xrange(height):
16         for x in xrange(width):
17             r, g, b = img.getpixel((x, y))
18             # print (r,g,b),
19             binresult += str(r%2) + str(g%2) # + str(b%2)
20         # print
21     for i in xrange(len(binresult) / 8):
```

```
22          v = binresult[8*i:8*(i+1)]
23          result += chr(int(v, 2))
24      open('embedded', 'wb').write(result)
25
26  if __name__ == '__main__':
27      main()
```

Il faut maintenant déterminer ce que l'on a extrait précisément :

```
$ file embedded
embedded: bzip2 compressed data, block size = 900k
$ tar tvjf embedded
bzip2: (stdin): trailing garbage after EOF ignored
-rw-r--r-- test/test     28755 2015-03-23 10:34 congratulations.gif
$ tar xjf embedded
$ rm -f embedded
```

Il s'agit donc d'une quatrième image (décidément), au format GIF [37] :

FIGURE 9 – src/stage6/congratulations.gif

Damn ! Bon. La première intuition évidente, c'est l'utilisation de plusieurs frames, pour former un GIF animé. Une autre piste crédible, c'est d'avoir caché un message dans la palette du fichier. L'idée de départ est donc d'utiliser hachoir à nouveau :

```
File
00000000  47 49 46 38 39 61 7c 02 da 01 e7 ff 00 00 00 00 00 00 00 00 00 0b 04 02 07 0a 06 0e 08 06 16  GIF89a|.....
00000020  0a 0b 29 08 07 2d 07 06 0e 10 0c 1d 0c 0c 14 0f 0e 24 0c 0b 38 07 06 12 13 10 16 17 15 2c 12 10  ..)..-.....
00000040  23 17 17 51 0b 08 1d 19 18 33 13 13 4b 0d 0d 39 12 11 19 1b 18 09 1e 2e 1f 1d 07 1c 1d 1b 43 13  $..8.....,..
00000060  12 3d 16 13 21 1e 1d 1e 20 1d 00 00 00 4b 17 16 2b 20 1c 2f 1f 1d 56 16 16 68 12 0c 61 14 0f 28  #..Q.....3..K..9.....
00000080  24 23 67 13 13 61 15 15 5b 17 13 24 26 23 77 11 0f 71 13 13 40 21 19 2c 2a 0a 00 00 00 82 12 15  C.
000000a0  83 13 10 2e 2c 21 2b 2c 2a 2f 2b 2a 8d 13 10 7e 18 14 96 12 13 a0 11 12 98 14 0e 13 35 52 35 32  .=..!... ....K..
000000c0  27 a9 12 10 36 31 30 3d 33 0d 32 33 31 74 22 1e be 0d 15 7f 20 1e 00 00 00 00 00 00 b6 14 11 c1  + ./..V..h..a..(
000000e0  11 0f 42 35 2f 36 38 35 b7 15 18 3c 38 36 56 34 18 68 33 2a 3e 40 3d 50 3f 12 44 3f 3e 44 41 35  $#g..a..[..
00000100  00 00 00 7f 34 19 00 00 00 00 00 00 ac 2d 15 aa 2d 2a 4b 47 45 4b 48 3c 47 49 46 00 00 00 4f 4a  $&#w..q..@!.,*.....
00000120  39 53 4e 4d 00 00 00 4f 51 4e 5e 53 0f 00 00 00 a7 3f 19 71 4d 3f 5a 54 42 7e 51 12 55 57 54 00  ....,!+,*/

address   name          type  size        data  description
          ../
0000000d.0 color[0]/    RGB   00000003.0      RGB color: Black
00000010.0 color[1]/    RGB   00000003.0      RGB color: Black
00000013.0 color[2]/    RGB   00000003.0      RGB color: Black
00000016.0 color[3]/    RGB   00000003.0      RGB color: #0B0402
00000019.0 color[4]/    RGB   00000003.0      RGB color: #070A06
0000001c.0 color[5]/    RGB   00000003.0      RGB color: #0E0806
0000001f.0 color[6]/    RGB   00000003.0      RGB color: #160A0B
```

37. http://en.wikipedia.org/wiki/GIF

On remarque que le GIF contient une seule image, ce n'est donc pas un GIF animé, donc il ne faut pas chercher de ce côté-là. Ce qui peut sembler étonnant par contre, c'est la présence de plusieurs occurences de la couleur noire dans la palette. Il peut être utile de faire quelques statistiques sur la palette contenue dans cette image. Notamment, on constate que la couleur noire est présente 54 fois dans la palettes (sur 256 slots), ce qui n'est pas commun - mais cela peut être dû à des slots non utilisés, non attribués. Mais quand on compte le nombre de couleurs réellement utilisées de la palette, on découvre que 254 couleurs de celle-ci sont utilisées. Donc au pire, il y a au moins 52 zones purement noires avec des index différents dans l'image. Il serait intéressant de plaquer une palette aléatoire sur cette image pour distinguer les occurences de noir. Typiquement, ce genre de camouflage ne pourrait pas se voir avec l'outil "pot de peinture" de gimp, puisque les noirs de la palette sont rigoureusement identiques. Nous allons donc implémenter un petit script pour automatiser cela :

Listing 22 – src/stage6/gifscramble.py

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

"Scramble the palette of a GIF picture"

__version__ = '0.1'

import PIL.Image
import random

def main():
    "main function"
    img     = PIL.Image.open('congratulations.gif')
    newpal = ''
    for _ in xrange(256 * 3):
        newpal += chr(random.randint(0, 255))
    img.palette.palette = newpal
    img.save('final.png')
    img.show()

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

Et le resultat ne se fait pas attendre :

FIGURE 10 – src/stage6/final.png

Hourra ! On constate bien la présence de l'adresse mail recherchée !

Après une rapide recherche, on peut aussi récupérer l'indice dans la palette de la couleur noire qui sert à former le contour, et on remarque qu'il s'agit de l'indice 2. On peut donc juste remplacer la couleur d'indice 2 dans la palette par du blanc :

Listing 23 – src/stage6/gifisolate.py

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

"Replace color[2] by white in a GIF palette"

__version__ = '0.1'

import PIL.Image

def main():
    "main function"
    img = PIL.Image.open('congratulations.gif')
    pal = img.palette.palette
    pal = pal[:2*3] + '\xFF'*3 + pal[3*3:]
    img.palette.palette = pal
    img.save('finalisolate.png')
    img.show()

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

Ce qui donne :

FIGURE 11 – src/stage6/finalisolate.png

Le résultat est donc :

`1713e7c1d0b750ccd4e002bb957aa799@challenge.sstic.org`

Envoyez vos spams !

A Miasm2 Pull Request

Listing 24 – src/miasm2/miasm2_analysis_gdbserver.py.patch

```
diff --git a/miasm2/analysis/gdbserver.py b/miasm2/analysis/gdbserver.py
index a930cc8..ad8238d 100644
--- a/miasm2/analysis/gdbserver.py
+++ b/miasm2/analysis/gdbserver.py
@@ -8,6 +8,7 @@ import logging
 from StringIO import StringIO
 import miasm2.analysis.debugging as debugging
 from miasm2.jitter.jitload import ExceptionHandle
+from collections import defaultdict


 class GdbServer(object):
@@ -429,3 +430,10 @@ class GdbServer_msp430(GdbServer):
         else:
             return sup_func(reg_name)

+
+class GdbServer_st20c4(GdbServer):
+    "Extend GdbServer for st20c4 purposes"
+    general_registers_order = ["Areg", "Breg", "Creg", "Iptr",
+                               "Status", "Wptr", "Tdesc", "IOreg",]
+    general_registers_size = defaultdict(lambda: 4)
+
```

Listing 25 – src/miasm2/miasm2_analysis_machine.py.patch

```
diff --git a/miasm2/analysis/machine.py b/miasm2/analysis/machine.py
index 778c5de..d1f4eb5 100644
--- a/miasm2/analysis/machine.py
+++ b/miasm2/analysis/machine.py
@@ -12,7 +12,7 @@ class Machine(object):
     __gdbserver = None    # GdbServer handler

     __available = ["arml", "armb", "armtl", "armtb", "sh4", "x86_16", "x86_32",
-                   "x86_64", "msp430", "mips32b", "mips32l"]
+                   "x86_64", "msp430", "mips32b", "mips32l", "st20c4"]


     def __init__(self, machine_name):
@@ -102,6 +102,14 @@ class Machine(object):
             jitter = jit.jitter_mips32l
             from miasm2.arch.mips32.ira import ir_a_mips32l as ira
             from miasm2.arch.mips32.sem import ir_mips32l as ir
+        elif machine_name == "st20c4":
+            from miasm2.arch.st20c4.disasm import dis_st20c4 as dis_engine
+            from miasm2.arch.st20c4 import arch, jit
+            mn = arch.mn_st20c4
+            jitter = jit.jitter_st20c4
+            from miasm2.arch.st20c4.ira import ir_a_st20c4 as ira
+            from miasm2.arch.st20c4.sem import ir_st20c4 as ir
+            from miasm2.analysis.gdbserver import GdbServer_st20c4 as gdbserver
         else:
             raise ValueError('Unknown machine: %s' % machine_name)
```

Listing 26 – src/miasm2/miasm2_arch_st20c4_arch.py.patch

```
diff --git a/opt/miasm2/miasm2/arch/st20c4/arch.py b/opt/miasm2/miasm2/arch/st20c4/arch.py
new file mode 100644
index 0000000..b448deb
--- /dev/null
+++ b/opt/miasm2/miasm2/arch/st20c4/arch.py
@@ -0,0 +1,210 @@
+#!/usr/bin/env python
+#-*- coding:utf-8 -*-
+
```

```
+from miasm2.core.cpu import cls_mn, instruction, str_int, m_arg, bs, base_expr
+from miasm2.expression.expression import ExprInt, ExprInt_fromsize, ExprMem
+from miasm2.arch.st20c4.regs import *
+import miasm2.arch.st20c4.regs as regs_module
+from collections import defaultdict
+from re import match
+from pyparsing import *
+from sys import stderr
+
+import logging
+log = logging.getLogger('arch_st20c4')
+hnd = logging.StreamHandler()
+hnd.setFormatter(logging.Formatter("[%(levelname)s]: %(message)s"))
+log.addHandler(hnd)
+log.setLevel(logging.CRITICAL)
+
+uint32_to_int32 = lambda x: x if x < 1 << 31 else x - (1 << 32)
+
+def GeneratePrefixSequence(value):
+    # prefix(mnemo, 0)
+    # = mnemo(0x0)
+
+    # prefix(mnemo, 0xFFFFFFB4)
+    # = prefix(nfix, 0x00000004) ; mnemo(0x4)
+    # = nfix(0x4) ; mnemo(0x4)
+
+    # prefix(mnemo, 0xFFFFFEFF)
+    # = prefix(nfix, 0x00000010) ; mnemo(0xF)
+    # = prefix(pfix, 0x1) ; nfix(0x0) ; mnemo(0xF)
+    # = pfix(0x1) ; nfix(0x0) ; mnemo(0xF)
+
+    def prefix(op, e):
+        if   0x0 <= e <= 0xF:
+             return [op + e]
+        elif e > 0xF:
+             return prefix(0x20,  e >> 4) + [op + (e & 0xF)]
+        elif e < 0x0:
+             return prefix(0x60, ~e >> 4) + [op + (e & 0xF)]
+    x = prefix(0, uint32_to_int32(value))
+    return x[:-1], x[-1]
+
+class additional_info:
+    prefixes = []
+
+class instruction_st20c4(instruction):
+    delayslot = 0
+    @staticmethod
+    def arg2str(e, pos = None):
+        return str(e)
+    def __str__(self):
+        return super(instruction_st20c4, self).__str__()
+    def splitflow(self):
+        if  self.name in ['CJ']:
+            return True
+        return False
+    def breakflow(self):
+        if  self.name in ['CJ', 'J', 'RET']:
+            return True
+        return False
+    def dstflow(self):
+        if  self.name in ['CJ', 'J']:
+            return True
+        return False
+    def dstflow2label(self, symbol_pool):
+        e  = self.args[0]
+        sz = len(self.additional_info.prefixes) + 1
+        ad = self.offset + sz + e.arg
+        l  = symbol_pool.getby_offset_create(ad)
+        r  = ExprId(l, e.size)
+        self.args[0] = r
```

```
+    def getdstflow(self, symbol_pool):
+        return [self.args[0]]
+    def is_subcall(self):
+        return False
+
+class mn_st20c4(cls_mn):
+    name = 'st20c4'
+    num  = 0
+    delayslot = 0
+    bintree = {}
+    all_mn  = []
+    all_mn_mode = defaultdict(list)
+    all_mn_name = defaultdict(list)
+    all_mn_inst = defaultdict(list)
+    regs        = regs_module
+    instruction = instruction_st20c4
+    pc = {None: Iptr}
+    sp = {None: Wptr}
+    @classmethod
+    def getpc(cls, attrib):
+        return Iptr
+    @classmethod
+    def getsp(cls, attrib):
+        return Wptr
+    @classmethod
+    def getmn(cls, name):
+        return name.upper()
+    @classmethod
+    def gen_modes(cls, subcls, name, bases, dct, fields):
+        dct['mode'] = None
+        return [(subcls, name, bases, dct, fields)]
+    def additional_info(self):
+        info = additional_info()
+        if hasattr(self, 'prefixes'):
+            info.prefixes = self.prefixes
+        return info
+    def dup_info(self, info):
+        if info is not None:
+            self.prefixes = info.prefixes
+    def add_pre_dis_info(self, pre_dis_info=None):
+        #print >> stderr, '[DEBUG]', 'add_pre_dis_info', pre_dis_info
+        if pre_dis_info is not None:
+            self.prefixes = pre_dis_info['prefixes']
+        return True
+    @classmethod
+    def pre_dis(cls, v, mode, offset):
+        o, p = 0, []
+        while True:
+            c = ord(v.getbytes(offset + o))
+            if  c >> 4 in [0x2, 0x6]:
+                p.append(c)
+                o += 1
+                continue
+            elif c >> 4 == 0xF:
+                o, p = 0, []
+            break
+        r = {'prefixes': p}, v, mode, offset + o, o
+        #print >> stderr, '[DEBUG]', 'predis', r
+        return r
+    def encodefields(self, fields):
+        v = ''
+        #print >> stderr, '[DEBUG]', 'encodefields!'
+        if hasattr(self, 'prefixes'):
+            for prefix in self.prefixes:
+                v += chr(prefix)
+        v += super(mn_st20c4, self).encodefields(fields)
+        return v
+    @classmethod
+    def fromstring(cls, s, mode):
+        c = super(mn_st20c4, cls).fromstring(s, mode)
```

```
+        if  c.name in ['J','LDLP','LDC','LDNLP','LDL','ADC','CJ','AJW','EQC','STL']:
+            p, v = GeneratePrefixSequence(c.args[0].arg.arg)
+            c.additional_info.prefixes = p
+            #c.args = [ExprInt_fromsize(4, v)]
+        return c
+
+def addop(name, fields):
+    type(name, (mn_st20c4,), {'fields': fields})
+
+class st20c4_instrdata(m_arg):
+    parser = base_expr
+    def decode(self, v):
+        #print >> stderr, '[DEBUG]', 'DECODE', v, self.parent.additional_info().prefixes
+        r = 0
+        for prefix in self.parent.additional_info().prefixes:
+            p1, p2 = prefix >> 4, prefix & 0xF
+            r += p2
+            if p1 == 0x6: r = (~r) & 0xFFFFFFFF
+            r = (r << 4) & 0xFFFFFFFF
+        self.expr = ExprInt_fromsize(32, r + v)
+        return True
+    def encode(self):
+        prefixes, remaining = GeneratePrefixSequence(self.expr.arg.arg)
+        self.parent.additional_info().prefixes = prefixes
+        #print >> stderr, '[DEBUG]', 'ENCODE', self.expr.arg.arg, (prefixes, remaining)
+        self.value = remaining
+        return True
+
+instrdata = bs(l=4, cls=(st20c4_instrdata,))
+
+# Primary instructions
+
+addop('j',      [bs('0000'), instrdata])                    #    0n = j
+addop('ldlp',   [bs('0001'), instrdata])                    #    1n = ldlp
+addop('ldc',    [bs('0100'), instrdata])                    #    4n = ldc
+addop('ldnlp',  [bs('0101'), instrdata])                    #    5n = ldnlp
+addop('ldl',    [bs('0111'), instrdata])                    #    7n = ldl
+addop('adc',    [bs('1000'), instrdata])                    #    8n = adc
+addop('call',   [bs('1001'), instrdata])                    #    9n = call
+addop('cj',     [bs('1010'), instrdata])                    #    An = cj
+addop('ajw',    [bs('1011'), instrdata])                    #    Bn = ajw
+addop('eqc',    [bs('1100'), instrdata])                    #    Cn = eqc
+addop('stl',    [bs('1101'), instrdata])                    #    Dn = stl
+
+# Secondary instructions
+
+addop('lb',     [               bs('11110001')])            #    F1 = lb
+addop('bsub',   [               bs('11110010')])            #    F2 = bsub
+addop('gcall',  [               bs('11110110')])            #    F6 = gcall
+addop('in',     [               bs('11110111')])            #    F7 = in
+addop('prod',   [               bs('11111000')])            #    F8 = prod
+addop('gt',     [               bs('11111001')])            #    F9 = gt
+addop('wsub',   [               bs('11111010')])            #    FA = wsub
+addop('out',    [               bs('11111011')])            #    FB = out
+addop('ldpi',   [bs('00100001'), bs('11111011')])           # 21 FB = ldpi
+addop('rem',    [bs('00100001'), bs('11111111')])           # 21 FF = rem
+addop('ret',    [bs('00100010'), bs('11110000')])           # 22 F0 = ret
+addop('xor',    [bs('00100011'), bs('11110011')])           # 23 F3 = xor
+addop('sb',     [bs('00100011'), bs('11111011')])           # 23 FB = sb
+addop('gajw',   [bs('00100011'), bs('11111100')])           # 23 FC = gajw
+addop('shr',    [bs('00100100'), bs('11110000')])           # 24 F0 = shr
+addop('shl',    [bs('00100100'), bs('11110001')])           # 24 F1 = shl
+addop('mint',   [bs('00100100'), bs('11110010')])           # 24 F2 = mint
+addop('and',    [bs('00100100'), bs('11110110')])           # 24 F6 = and
+addop('dup',    [bs('00100101'), bs('11111010')])           # 25 FA = dup
+addop('ssub',   [bs('00101100'), bs('11110001')])           # 2C F1 = ssub
+
```

Listing 27 – src/miasm2/miasm2_arch_st20c4_disasm.py.patch

```
diff --git a/opt/miasm2/miasm2/arch/st20c4/disasm.py b/opt/miasm2/miasm2/arch/st20c4/disasm.py
new file mode 100644
index 0000000..da1a1b4
--- /dev/null
+++ b/opt/miasm2/miasm2/arch/st20c4/disasm.py
@@ -0,0 +1,10 @@
+#!/usr/bin/env python
+#-*- coding:utf-8 -*-
+
+from miasm2.core.asmbloc import disasmEngine
+from miasm2.arch.st20c4.arch import mn_st20c4
+
+class dis_st20c4(disasmEngine):
+    def __init__(self, bs=None, **kwargs):
+        super(dis_st20c4, self).__init__(mn_st20c4, None, bs, **kwargs)
+
```

Listing 28 – src/miasm2/miasm2_arch_st20c4____init___.py.patch

```
diff --git a/opt/miasm2/miasm2/arch/st20c4/__init__.py b/opt/miasm2/miasm2/arch/st20c4/__init__.py
new file mode 100644
index 0000000..e69de29
```

Listing 29 – src/miasm2/miasm2_arch_st20c4_ira.py.patch

```
diff --git a/opt/miasm2/miasm2/arch/st20c4/ira.py b/opt/miasm2/miasm2/arch/st20c4/ira.py
new file mode 100644
index 0000000..647e4b7
--- /dev/null
+++ b/opt/miasm2/miasm2/arch/st20c4/ira.py
@@ -0,0 +1,17 @@
+#!/usr/bin/env python
+#-*- coding:utf-8 -*-
+
+from miasm2.expression.expression import *
+from miasm2.ir.ir import ir, irbloc
+from miasm2.ir.analysis import ira
+from miasm2.arch.st20c4.sem import ir_st20c4
+from miasm2.arch.st20c4.regs import *
+
+class ir_a_st20c4_base(ir_st20c4, ira):
+    def __init__(self, symbol_pool=None):
+        ir_st20c4.__init__(self, symbol_pool)
+
+class ir_a_st20c4(ir_a_st20c4_base):
+    def __init__(self, symbol_pool=None):
+        ir_a_st20c4_base.__init__(self, symbol_pool)
+
```

Listing 30 – src/miasm2/miasm2_arch_st20c4_jit.py.patch

```
diff --git a/opt/miasm2/miasm2/arch/st20c4/jit.py b/opt/miasm2/miasm2/arch/st20c4/jit.py
new file mode 100644
index 0000000..14b6eb0
--- /dev/null
+++ b/opt/miasm2/miasm2/arch/st20c4/jit.py
@@ -0,0 +1,32 @@
+#!/usr/bin/env python
+#-*- coding:utf-8 -*-
+
+from miasm2.jitter.jitload import jitter
+from miasm2.jitter.csts    import *
+from miasm2.core           import asmbloc
+from miasm2.core.utils     import *
+
+import logging
+log = logging.getLogger('jit_st20c4')
+hnd = logging.StreamHandler()
```

```
+hnd.setFormatter(logging.Formatter("[%(levelname)s]: %(message)s"))
+log.addHandler(hnd)
+log.setLevel(logging.CRITICAL)
+
+class jitter_st20c4(jitter):
+    def __init__(self, *args, **kwargs):
+        from miasm2.arch.st20c4.sem import ir_st20c4
+        sp = asmbloc.asm_symbol_pool()
+        jitter.__init__(self, ir_st20c4(sp), *args, **kwargs)
+        self.vm.set_little_endian()
+        self.ir_arch.jit_pc = self.ir_arch.arch.regs.Iptr
+        self.stack_base = 0x20000000
+        self.stack_size = 0x2000
+    def init_run(self, *args, **kwargs):
+        jitter.init_run(self, *args, **kwargs)
+        self.cpu.Iptr = self.pc
+    def init_stack(self):
+        self.vm.add_memory_page(self.stack_base - self.stack_size,
+            PAGE_READ | PAGE_WRITE, '\xAA' * self.stack_size)
+        setattr(self.cpu, self.arch.getsp(self.attrib).name, self.stack_base)
+
```

Listing 31 – src/miasm2/miasm2_arch_st20c4_regs.py.patch

```
diff --git a/opt/miasm2/miasm2/arch/st20c4/regs.py b/opt/miasm2/miasm2/arch/st20c4/regs.py
new file mode 100644
index 0000000..c321fd1
--- /dev/null
+++ b/opt/miasm2/miasm2/arch/st20c4/regs.py
@@ -0,0 +1,44 @@
+#!/usr/bin/env python
+#-*- coding:utf-8 -*-
+
+from miasm2.expression.expression import ExprId
+
+Areg   = ExprId('Areg',   32)
+Breg   = ExprId('Breg',   32)
+Creg   = ExprId('Creg',   32)
+Iptr   = ExprId('Iptr',   32)
+Status = ExprId('Status', 32)
+Wptr   = ExprId('Wptr',   32)
+Tdesc  = ExprId('Tdesc',  32)
+IOreg  = ExprId('IOreg',  32)
+
+exception_flags = ExprId('exception_flags', 32)
+
+Areg_init   = ExprId('Areg_init',   32)
+Breg_init   = ExprId('Breg_init',   32)
+Creg_init   = ExprId('Creg_init',   32)
+Iptr_init   = ExprId('Iptr_init',   32)
+Status_init = ExprId('Status_init', 32)
+Wptr_init   = ExprId('Wptr_init',   32)
+Tdesc_init  = ExprId('Tdesc_init',  32)
+IOreg_init  = ExprId('IOreg_init',  32)
+
+exception_flags_init = ExprId('exception_flags_init', 32)
+
+all_regs_ids = [
+    Areg, Breg, Creg, Iptr,
+    Status, Wptr, Tdesc, IOreg,
+    exception_flags,
+]
+
+all_regs_ids_init = [
+    Areg_init, Breg_init, Creg_init, Iptr_init,
+    Status_init, Wptr_init, Tdesc_init, IOreg_init,
+    exception_flags_init,
+]
+
+all_regs_ids_no_alias = all_regs_ids
```

```
+all_regs_ids_byname    = {x.name: x for x in all_regs_ids}
+
+regs_flt_expr = []
+
```

Listing 32 – src/miasm2/miasm2_arch_st20c4_sem.py.patch

```
diff --git a/opt/miasm2/miasm2/arch/st20c4/sem.py b/opt/miasm2/miasm2/arch/st20c4/sem.py
new file mode 100644
index 0000000..94357a2
--- /dev/null
+++ b/opt/miasm2/miasm2/arch/st20c4/sem.py
@@ -0,0 +1,291 @@
+#!/usr/bin/env python
+#-*- coding:utf-8 -*-
+
+from sys import stderr
+from miasm2.expression.expression import *
+from miasm2.arch.st20c4.regs import *
+from miasm2.arch.st20c4.arch import mn_st20c4
+from miasm2.ir.ir import ir
+
+nwords = lambda n: ExprInt32(4) * n
+
+ST20C4_MOSTNEG = ExprInt32(0x80000000)
+
+ST20C4_EXCEPTIONFLAG_IN  = 1 << 1
+ST20C4_EXCEPTIONFLAG_OUT = 1 << 2
+
+# Primary instructions
+
+def mnemo_j(ir, instr, n):
+    # page 95
+    dst = n
+    e = [
+        ExprAff(Iptr,      dst),
+        ExprAff(ir.IRDst, dst),
+    ]
+    return e, []
+
+def mnemo_ldlp(ir, instr, n):
+    # page 106
+    e = [
+        ExprAff(Areg, Wptr + nwords(n)),
+        ExprAff(Breg, Areg),
+        ExprAff(Creg, Breg),
+    ]
+    return e, []
+
+def mnemo_ldc(ir, instr, n):
+    # page 99
+    e = [
+        ExprAff(Areg, n),
+        ExprAff(Breg, Areg),
+        ExprAff(Creg, Breg),
+    ]
+    return e, []
+
+def mnemo_ldnlp(ir, instr, n):
+    # page 109
+    e = [
+        ExprAff(Areg, Areg + nwords(n))
+    ]
+    return e, []
+
+def mnemo_ldl(ir, instr, n):
+    # page 105
+    e = [
+        ExprAff(Areg, ExprMem(Wptr + nwords(n))),
+        ExprAff(Breg, Areg),
```

```
+        ExprAff(Creg, Breg),
+    ]
+    return e, []
+
+def mnemo_adc(ir, instr, n):
+    # page 32
+    e = [
+        ExprAff(Areg, Areg + n),
+    ]
+    return e, []
+
+def mnemo_call(ir, instr, n):
+    # page 44
+    e = [
+        ExprAff(Wptr, Wptr   - nwords(4)),
+        ExprAff(ExprMem(Wptr - nwords(4)), Iptr + ExprInt32(2)),
+        ExprAff(ExprMem(Wptr - nwords(3)), Areg),
+        ExprAff(ExprMem(Wptr - nwords(2)), Breg),
+        ExprAff(ExprMem(Wptr - nwords(1)), Creg),
+        ExprAff(Iptr, Iptr + ExprInt32(2) + nwords(n)),
+        ExprAff(Areg, Iptr + ExprInt32(2)),
+    ]
+    return e, []
+
+def mnemo_cj(ir, instr, n):
+    # page 52
+    dst = ExprCond(Areg,
+        ExprId(ir.get_next_label(instr), 32),
+        n,
+    )
+    e = [
+        ExprAff(Iptr,      dst),
+        ExprAff(ir.IRDst, dst),
+        ExprAff(Areg, ExprCond(Areg, Breg, Areg)),
+        ExprAff(Breg, ExprCond(Areg, Creg, Breg)),
+    ]
+    return e, []
+
+def mnemo_ajw(ir, instr, n):
+    # page 34
+    e = [
+        ExprAff(Wptr, Wptr + nwords(n)),
+    ]
+    return e, []
+
+def mnemo_eqc(ir, instr, n):
+    # page 81
+    e = [
+        ExprAff(Areg, ExprCond(Areg - n, ExprInt32(0), ExprInt32(1))),
+    ]
+    return e, []
+
+def mnemo_stl(ir, instr, n):
+    # page 170
+    e = [
+        ExprAff(ExprMem(Wptr + nwords(n)), Areg),
+        ExprAff(Areg, Breg),
+        ExprAff(Breg, Creg),
+    ]
+    return e, []
+
+# Secondary instructions
+
+def mnemo_lb(ir, instr):
+    # page 97
+    e = [
+        ExprAff(Areg, ExprCompose([
+            (ExprInt_fromsize(24, 0), 0, 24),
+            (ExprMem(Areg, 8)[:8], 24, 32)])),
+    ]
```

```
+    return e, []
+
+def mnemo_bsub(ir, instr):
+    # page 43
+    e = [
+        ExprAff(Areg, Areg + Breg),
+        ExprAff(Breg, Creg),
+    ]
+    return e, []
+
+def mnemo_gcall(ir, instr):
+    # page 85
+    e = [
+        ExprAff(Iptr, Areg),
+        ExprAff(Areg, Iptr + ExprInt32(1)),
+    ]
+    return e, []
+
+def mnemo_in(ir, instr):
+    # page 90
+    e = [
+        ExprAff(exception_flags, ExprInt32(ST20C4_EXCEPTIONFLAG_IN)),
+    ]
+    return e, []
+
+def mnemo_prod(ir, instr):
+    # page 142
+    e = [
+        ExprAff(Areg, Areg * Breg),
+        ExprAff(Breg, Creg),
+    ]
+    return e, []
+
+def mnemo_gt(ir, instr):
+    # page 88
+    e = [
+        ExprAff(Areg, ExprCond(Breg - Areg, ExprInt32(1), ExprInt32(0))),
+        ExprAff(Breg, Creg),
+    ]
+    return e, []
+
+def mnemo_wsub(ir, instr):
+    # page 199
+    e = [
+        ExprAff(Areg, Areg + nwords(Breg)),
+        ExprAff(Breg, Creg),
+    ]
+    return e, []
+
+def mnemo_out(ir, instr):
+    # page 137
+    e = [
+        ExprAff(exception_flags, ExprInt32(ST20C4_EXCEPTIONFLAG_OUT)),
+    ]
+    return e, []
+
+def mnemo_ldpi(ir, instr):
+    # page 110
+    e = [
+        ExprAff(Areg, Areg + Iptr + ExprInt32(2)),
+    ]
+    return e, []
+
+def mnemo_rem(ir, instr):
+    # page 144
+    e = [
+        ExprAff(Areg, Areg % Breg),
+        ExprAff(Breg, Creg),
+    ]
+    return e, []
```

```
+
+def mnemo_ret(ir, instr):
+    # page 147
+    e = [
+        ExprAff(Iptr, ExprMem(Wptr)),
+        ExprAff(Wptr, Wptr + nwords(4)),
+    ]
+    return e, []
+
+def mnemo_xor(ir, instr):
+    # page 203
+    e = [
+        ExprAff(Areg, Areg ^ Breg),
+        ExprAff(Breg, Creg),
+    ]
+    return e, []
+
+def mnemo_sb(ir, instr):
+    # page 156
+    e = [
+        ExprAff(ExprMem(Areg, 8), Breg[:8]),
+        ExprAff(Areg, Creg),
+    ]
+    return e, []
+
+def mnemo_gajw(ir, instr):
+    # page 84
+    e = [
+        ExprAff(ExprMem(Wptr), Areg),
+        ExprAff(Areg, ExprMem(Wptr)),
+    ]
+    return e, []
+
+def mnemo_shr(ir, instr):
+    # page 161
+    e = [
+        ExprAff(Areg, Breg >> Areg),
+        ExprAff(Breg, Creg),
+    ]
+    return e, []
+
+def mnemo_shl(ir, instr):
+    # page 160
+    e = [
+        ExprAff(Areg, Breg << Areg),
+        ExprAff(Breg, Creg),
+    ]
+    return e, []
+
+def mnemo_mint(ir, instr):
+    # page 126
+    e = [
+        ExprAff(Areg, ST20C4_MOSTNEG),
+        ExprAff(Breg, Areg),
+        ExprAff(Creg, Breg),
+    ]
+    return e, []
+
+def mnemo_and(ir, instr):
+    # page 38
+    e = [
+        ExprAff(Areg, Areg & Breg),
+        ExprAff(Breg, Creg),
+    ]
+    return e, []
+
+def mnemo_dup(ir, instr):
+    # page 76
+    e = [
+        ExprAff(Breg, Areg),
```

```
+        ExprAff(Creg, Breg),
+    ]
+    return e, []
+
+def mnemo_ssub(ir, instr):
+    # page 165
+    e = [
+        ExprAff(Areg, Areg + ExprInt32(2) * Breg),
+        ExprAff(Breg, Creg),
+    ]
+    return e, []
+
+
+class ir_st20c4(ir):
+    def __init__(self, symbol_pool=None):
+        ir.__init__(self, mn_st20c4, None, symbol_pool)
+        self.pc = Iptr
+        self.sp = Wptr
+        self.IRDst = ExprId('IRDst', 32)
+    def mod_pc(self, instr, instr_ir, extra_ir):
+        pass
+    def get_ir(self, instr):
+        return globals()['mnemo_' + instr.name.lower()](self, instr, *instr.args)
+
```

Listing 33 – src/miasm2/miasm2_ir_ir2C.py.patch

```
diff --git a/miasm2/ir/ir2C.py b/miasm2/ir/ir2C.py
index 4c9dff8..5685cea 100644
--- a/miasm2/ir/ir2C.py
+++ b/miasm2/ir/ir2C.py
@@ -339,7 +339,8 @@ def Expr2C(ir_arch, l, exprs, gen_exception_code=False):
                     code_exception_post_instr_noautomod % (s1, s2))

         if fetch_mem:
-            if l.additional_info.except_on_instr:
+            if (hasattr(l.additional_info, 'except_on_instr') and
+                l.additional_info.except_on_instr):
                 offset = l.offset
             else:
                 offset = l.offset + l.l
@@ -390,10 +391,12 @@ def ir2C(ir_arch, irbloc, lbl_done,
             out.append([pre_instr_test_exception % (s1)])
             lbl_done.add(l.offset)

-            if log_regs:
+            if (type(log_regs) == bool and log_regs or
+                hasattr(log_regs, '__call__') and log_regs(l)):
                 out.append([r'dump_gpregs(vmcpu);'])

-            if log_mn:
+            if (type(log_mn)   == bool and log_mn   or
+                hasattr(log_mn,   '__call__') and log_mn(l)):
                 out.append(['printf("%.8X %s\\n");' % (l.offset, str(l))])
         # print l
         # gen pc update
```

Listing 34 – src/miasm2/miasm2_jitter_arch_JitCore_st20c4.c.patch

```
diff --git a/opt/miasm2/miasm2/jitter/arch/JitCore_st20c4.c b/opt/miasm2/miasm2/jitter/arch/JitCore_st20c4.c
new file mode 100644
index 0000000..1132252
--- /dev/null
+++ b/opt/miasm2/miasm2/jitter/arch/JitCore_st20c4.c
@@ -0,0 +1,367 @@
+#include <Python.h>
+#include "JitCore.h"
+#include "structmember.h"
+#include <stdint.h>
```

```
+#include <inttypes.h>
+#include "JitCore_st20c4.h"
+
+#define RAISE(errtype, msg) {PyObject* p; p = PyErr_Format( errtype, msg ); return p;}
+
+typedef struct _reg_dict{
+    char* name;
+    unsigned long offset;
+} reg_dict;
+
+
+#define PyGetInt(item, value)                                          \
+       if (PyInt_Check(item)){                                         \
+               value = (uint64_t)PyInt_AsLong(item);                   \
+       }                                                               \
+       else if (PyLong_Check(item)){                                   \
+               value = (uint64_t)PyLong_AsUnsignedLongLong(item);      \
+       }                                                               \
+       else{                                                           \
+               RAISE(PyExc_TypeError,"arg must be int");               \
+       }                                                               \
+
+reg_dict gpreg_dict[] = { {.name = "Areg", .offset = offsetof(vm_cpu_t, Areg)},
+                          {.name = "Breg", .offset = offsetof(vm_cpu_t, Breg)},
+                          {.name = "Creg", .offset = offsetof(vm_cpu_t, Creg)},
+                          {.name = "Iptr", .offset = offsetof(vm_cpu_t, Iptr)},
+                          {.name = "Status", .offset = offsetof(vm_cpu_t, Status)},
+                          {.name = "Wptr", .offset = offsetof(vm_cpu_t, Wptr)},
+                          {.name = "Tdesc", .offset = offsetof(vm_cpu_t, Tdesc)},
+                          {.name = "IOreg", .offset = offsetof(vm_cpu_t, IOreg)},
+
+};
+
+/************************* JitCpu object ************************/
+
+typedef struct {
+       PyObject_HEAD
+       PyObject *cpu; /* cpu */
+       vm_cpu_t vmcpu;
+} JitCpu;
+
+
+
+#define get_reg(reg)  do {                                             \
+               o = PyLong_FromUnsignedLongLong((uint64_t)self->vmcpu.reg); \
+               PyDict_SetItemString(dict, #reg, o);                    \
+               Py_DECREF(o);                                           \
+       } while(0);
+
+
+
+PyObject* cpu_get_gpreg(JitCpu* self)
+{
+    PyObject *dict = PyDict_New();
+    PyObject *o;
+
+    get_reg(Areg);
+    get_reg(Breg);
+    get_reg(Creg);
+    get_reg(Iptr);
+    get_reg(Status);
+    get_reg(Wptr);
+    get_reg(Tdesc);
+    get_reg(IOreg);
+
+    return dict;
+}
+
+#define get_reg_off(reg)  do {                                         \
+               o = PyLong_FromUnsignedLongLong((uint64_t)offsetof(vm_cpu_t, reg)); \
```

```
+                PyDict_SetItemString(dict, #reg, o);                    \
+                Py_DECREF(o);                                           \
+        } while(0);
+
+
+PyObject* get_gpreg_offset_all(void)
+{
+    PyObject *dict = PyDict_New();
+    PyObject *o;
+    get_reg_off(exception_flags);
+    get_reg_off(exception_flags_new);
+    get_reg_off(Areg);
+    get_reg_off(Breg);
+    get_reg_off(Creg);
+    get_reg_off(Iptr);
+    get_reg_off(Status);
+    get_reg_off(Wptr);
+    get_reg_off(Tdesc);
+    get_reg_off(IOreg);
+    get_reg_off(Areg_new);
+    get_reg_off(Breg_new);
+    get_reg_off(Creg_new);
+    get_reg_off(Iptr_new);
+    get_reg_off(Status_new);
+    get_reg_off(Wptr_new);
+    get_reg_off(Tdesc_new);
+    get_reg_off(IOreg_new);
+    get_reg_off(pfmem32_0);
+    return dict;
+}
+
+
+PyObject* _vm_set_gpreg(JitCpu* self, PyObject *dict)
+{
+    PyObject *d_key, *d_value = NULL;
+    Py_ssize_t pos = 0;
+    uint64_t val;
+    unsigned int i, found;
+
+    if(!PyDict_Check(dict))
+            RAISE(PyExc_TypeError, "arg must be dict");
+    while(PyDict_Next(dict, &pos, &d_key, &d_value)){
+            if(!PyString_Check(d_key))
+                    RAISE(PyExc_TypeError, "key must be str");
+
+            PyGetInt(d_value, val);
+
+
+            found = 0;
+            for (i=0; i < sizeof(gpreg_dict)/sizeof(reg_dict); i++){
+                    if (strcmp(PyString_AsString(d_key), gpreg_dict[i].name))
+                            continue;
+                    *((uint32_t*)((char*)&(self->vmcpu)) + gpreg_dict[i].offset)) = val;
+                    found = 1;
+                    break;
+            }
+
+            if (found)
+                    continue;
+            fprintf(stderr, "unkown key: %s\n", PyString_AsString(d_key));
+            RAISE(PyExc_ValueError, "unkown reg");
+    }
+    return NULL;
+}
+
+PyObject* cpu_set_gpreg(JitCpu* self, PyObject *args)
+{
+        PyObject* dict;
+        if (!PyArg_ParseTuple(args, "O", &dict))
+                return NULL;
```

```
+        _vm_set_gpreg(self, dict);
+        Py_INCREF(Py_None);
+        return Py_None;
+}
+
+
+PyObject* cpu_set_exception(JitCpu* self, PyObject* args)
+{
+        PyObject *item1;
+        uint64_t i;
+
+        if (!PyArg_ParseTuple(args, "O", &item1))
+                return NULL;
+
+        PyGetInt(item1, i);
+
+        self->vmcpu.exception_flags = i;
+        Py_INCREF(Py_None);
+        return Py_None;
+}
+
+PyObject* cpu_get_exception(JitCpu* self, PyObject* args)
+{
+        return PyLong_FromUnsignedLongLong((uint64_t)self->vmcpu.exception_flags);
+}
+
+
+PyObject * cpu_init_regs(JitCpu* self)
+{
+        memset(&self->vmcpu, 0, sizeof(vm_cpu_t));
+
+        Py_INCREF(Py_None);
+        return Py_None;
+
+}
+
+void dump_gpregs(vm_cpu_t* vmcpu)
+{
+
+    printf("Areg    %.8"PRIX32" Breg  %.8"PRIX32" Creg   %.8"PRIX32" Iptr   %.8"PRIX32"\n",
+                vmcpu->Areg,    vmcpu->Breg, vmcpu->Creg,  vmcpu->Iptr);
+    printf("Status  %.8"PRIX32" Wptr  %.8"PRIX32" Tdesc  %.8"PRIX32" IOreg  %.8"PRIX32"\n",
+               vmcpu->Status, vmcpu->Wptr, vmcpu->Tdesc, vmcpu->IOreg);
+}
+
+
+PyObject * cpu_dump_gpregs(JitCpu* self, PyObject* args)
+{
+        vm_cpu_t* vmcpu;
+
+        vmcpu = &self->vmcpu;
+        dump_gpregs(vmcpu);
+        Py_INCREF(Py_None);
+        return Py_None;
+}
+
+
+
+static void
+JitCpu_dealloc(JitCpu* self)
+{
+    self->ob_type->tp_free((PyObject*)self);
+}
+
+
+static PyObject *
+JitCpu_new(PyTypeObject *type, PyObject *args, PyObject *kwds)
+{
+    JitCpu *self;
+
```

```
+    self = (JitCpu *)type->tp_alloc(type, 0);
+    return (PyObject *)self;
+}
+
+static PyObject *
+JitCpu_get_cpu(JitCpu *self, void *closure)
+{
+       return PyLong_FromUnsignedLongLong((uint64_t)&(self->vmcpu));
+}
+
+static int
+JitCpu_set_cpu(JitCpu *self, PyObject *value, void *closure)
+{
+       PyErr_SetString(PyExc_TypeError, "immutable cpu");
+       return -1;
+}
+
+static PyMemberDef JitCpu_members[] = {
+    {NULL}  /* Sentinel */
+};
+
+static PyMethodDef JitCpu_methods[] = {
+       {"init_regs", (PyCFunction)cpu_init_regs, METH_NOARGS,
+        "X"},
+       {"dump_gpregs", (PyCFunction)cpu_dump_gpregs, METH_NOARGS,
+        "X"},
+       {"get_gpreg", (PyCFunction)cpu_get_gpreg, METH_NOARGS,
+        "X"},
+       {"set_gpreg", (PyCFunction)cpu_set_gpreg, METH_VARARGS,
+        "X"},
+       {"get_exception", (PyCFunction)cpu_get_exception, METH_VARARGS,
+        "X"},
+       {"set_exception", (PyCFunction)cpu_set_exception, METH_VARARGS,
+        "X"},
+       {NULL}  /* Sentinel */
+};
+
+static int
+JitCpu_init(JitCpu *self, PyObject *args, PyObject *kwds)
+{
+       return 0;
+}
+
+getset_reg_u32(Areg);
+getset_reg_u32(Breg);
+getset_reg_u32(Creg);
+getset_reg_u32(Iptr);
+getset_reg_u32(Status);
+getset_reg_u32(Wptr);
+getset_reg_u32(Tdesc);
+getset_reg_u32(IOreg);
+
+
+
+
+static PyGetSetDef JitCpu_getseters[] = {
+    {"cpu",
+     (getter)JitCpu_get_cpu, (setter)JitCpu_set_cpu,
+     "first name",
+     NULL},
+
+
+    {"Areg" ,   (getter)JitCpu_get_Areg   , (setter)JitCpu_set_Areg   , "Areg" ,   NULL},
+    {"Breg" ,   (getter)JitCpu_get_Breg   , (setter)JitCpu_set_Breg   , "Breg" ,   NULL},
+    {"Creg" ,   (getter)JitCpu_get_Creg   , (setter)JitCpu_set_Creg   , "Creg" ,   NULL},
+    {"Iptr" ,   (getter)JitCpu_get_Iptr   , (setter)JitCpu_set_Iptr   , "Iptr" ,   NULL},
+    {"Status" , (getter)JitCpu_get_Status , (setter)JitCpu_set_Status , "Status" , NULL},
+    {"Wptr" ,   (getter)JitCpu_get_Wptr   , (setter)JitCpu_set_Wptr   , "Wptr" ,   NULL},
+    {"Tdesc" ,  (getter)JitCpu_get_Tdesc  , (setter)JitCpu_set_Tdesc  , "Tdesc" ,  NULL},
+    {"IOreg" ,  (getter)JitCpu_get_IOreg  , (setter)JitCpu_set_IOreg  , "IOreg" ,  NULL},
```

```
+
+    {NULL}  /* Sentinel */
+};
+
+
+
+static PyTypeObject JitCpuType = {
+    PyObject_HEAD_INIT(NULL)
+    0,                         /*ob_size*/
+    "JitCore_st20c4.JitCpu",      /*tp_name*/
+    sizeof(JitCpu),            /*tp_basicsize*/
+    0,                         /*tp_itemsize*/
+    (destructor)JitCpu_dealloc,/*tp_dealloc*/
+    0,                         /*tp_print*/
+    0,                         /*tp_getattr*/
+    0,                         /*tp_setattr*/
+    0,                         /*tp_compare*/
+    0,                         /*tp_repr*/
+    0,                         /*tp_as_number*/
+    0,                         /*tp_as_sequence*/
+    0,                         /*tp_as_mapping*/
+    0,                         /*tp_hash */
+    0,                         /*tp_call*/
+    0,                         /*tp_str*/
+    0,                         /*tp_getattro*/
+    0,                         /*tp_setattro*/
+    0,                         /*tp_as_buffer*/
+    Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE, /*tp_flags*/
+    "JitCpu objects",          /* tp_doc */
+    0,                        /* tp_traverse */
+    0,                        /* tp_clear */
+    0,                        /* tp_richcompare */
+    0,                        /* tp_weaklistoffset */
+    0,                        /* tp_iter */
+    0,                        /* tp_iternext */
+    JitCpu_methods,            /* tp_methods */
+    JitCpu_members,            /* tp_members */
+    JitCpu_getseters,          /* tp_getset */
+    0,                         /* tp_base */
+    0,                         /* tp_dict */
+    0,                         /* tp_descr_get */
+    0,                         /* tp_descr_set */
+    0,                         /* tp_dictoffset */
+    (initproc)JitCpu_init,     /* tp_init */
+    0,                         /* tp_alloc */
+    JitCpu_new,                /* tp_new */
+};
+
+
+
+static PyMethodDef JitCore_st20c4_Methods[] = {
+
+       /*
+
+       */
+       {"get_gpreg_offset_all", (PyCFunction)get_gpreg_offset_all, METH_NOARGS},
+       {NULL, NULL, 0, NULL}        /* Sentinel */
+
+};
+
+static PyObject *JitCore_st20c4_Error;
+
+PyMODINIT_FUNC
+initJitCore_st20c4(void)
+{
+    PyObject *m;
+
+    if (PyType_Ready(&JitCpuType) < 0)
+       return;
+
```

```
+    m = Py_InitModule("JitCore_st20c4", JitCore_st20c4_Methods);
+    if (m == NULL)
+            return;
+
+    JitCore_st20c4_Error = PyErr_NewException("JitCore_st20c4.error", NULL, NULL);
+    Py_INCREF(JitCore_st20c4_Error);
+    PyModule_AddObject(m, "error", JitCore_st20c4_Error);
+
+    Py_INCREF(&JitCpuType);
+    PyModule_AddObject(m, "JitCpu", (PyObject *)&JitCpuType);
+
+}
+
```

Listing 35 – src/miasm2/miasm2_jitter_arch_JitCore_st20c4.h.patch

```
diff --git a/opt/miasm2/miasm2/jitter/arch/JitCore_st20c4.h b/opt/miasm2/miasm2/jitter/arch/JitCore_st20c4.h
new file mode 100644
index 0000000..af1bc19
--- /dev/null
+++ b/opt/miasm2/miasm2/jitter/arch/JitCore_st20c4.h
@@ -0,0 +1,35 @@
+
+typedef struct {
+        uint32_t exception_flags;
+        uint32_t exception_flags_new;
+
+        /* gpregs */
+        uint32_t Areg;
+        uint32_t Breg;
+        uint32_t Creg;
+        uint32_t Iptr;
+        uint32_t Status;
+        uint32_t Wptr;
+        uint32_t Tdesc;
+        uint32_t IOreg;
+
+         uint32_t Areg_new;
+         uint32_t Breg_new;
+         uint32_t Creg_new;
+         uint32_t Iptr_new;
+         uint32_t Status_new;
+         uint32_t Wptr_new;
+         uint32_t Tdesc_new;
+         uint32_t IOreg_new;
+
+        /* eflag */
+
+         uint32_t pfmem32_0;
+
+        uint32_t segm_base[0x10000];
+
+}vm_cpu_t;
+
+//#define RETURN_PC return PyLong_FromUnsignedLongLong(vmcpu->PC);
+#define RETURN_PC return BlockDst;
+
```

Listing 36 – src/miasm2/miasm2_jitter_jitcore_llvm.py.patch

```
diff --git a/miasm2/jitter/jitcore_llvm.py b/miasm2/jitter/jitcore_llvm.py
index acf91d1..ed31809 100644
--- a/miasm2/jitter/jitcore_llvm.py
+++ b/miasm2/jitter/jitcore_llvm.py
@@ -14,7 +14,9 @@ class JitCore_LLVM(jitcore.JitCore):
     arch_dependent_libs = {"x86": "JitCore_x86.so",
                            "arm": "JitCore_arm.so",
                            "msp430": "JitCore_msp430.so",
-                           "mips32": "JitCore_mips32.so"}
```

```
+                               "mips32": "JitCore_mips32.so",
+                               "st20c4": "JitCore_st20c4.so",
+    }

     def __init__(self, ir_arch, bs=None):
         super(JitCore_LLVM, self).__init__(ir_arch, bs)
```

Listing 37 – src/miasm2/miasm2_jitter_jitload.py.patch

```
diff --git a/miasm2/jitter/jitload.py b/miasm2/jitter/jitload.py
index 97fd3c8..3646e71 100644
--- a/miasm2/jitter/jitload.py
+++ b/miasm2/jitter/jitload.py
@@ -190,6 +190,8 @@ class jitter:
             from miasm2.jitter.arch import JitCore_msp430 as jcore
         elif arch_name == "mips32":
             from miasm2.jitter.arch import JitCore_mips32 as jcore
+        elif arch_name == "st20c4":
+            from miasm2.jitter.arch import JitCore_st20c4 as jcore
         else:
             raise ValueError("unsupported jit arch!")
```

Listing 38 – src/miasm2/setup.py.patch

```
diff --git a/setup.py b/setup.py
index 8c2b100..09b9e3f 100755
--- a/setup.py
+++ b/setup.py
@@ -13,6 +13,7 @@ def buil_all():
                 'miasm2/arch/msp430',
                 'miasm2/arch/sh4',
                 'miasm2/arch/mips32',
+                'miasm2/arch/st20c4',
                 'miasm2/core',
                 'miasm2/expression',
                 'miasm2/ir',
@@ -39,6 +40,9 @@ def buil_all():
         Extension("miasm2.jitter.arch.JitCore_mips32",
                   ["miasm2/jitter/arch/JitCore.c",
                    "miasm2/jitter/arch/JitCore_mips32.c"]),
+        Extension("miasm2.jitter.arch.JitCore_st20c4",
+                  ["miasm2/jitter/arch/JitCore.c",
+                   "miasm2/jitter/arch/JitCore_st20c4.c"]),
         Extension("miasm2.jitter.Jitllvm",
                   ["miasm2/jitter/Jitllvm.c"]),
         ]
@@ -59,6 +63,9 @@ def buil_all():
         Extension("miasm2.jitter.arch.JitCore_mips32",
                   ["miasm2/jitter/arch/JitCore.c",
                    "miasm2/jitter/arch/JitCore_mips32.c"]),
+        Extension("miasm2.jitter.arch.JitCore_st20c4",
+                  ["miasm2/jitter/arch/JitCore.c",
+                   "miasm2/jitter/arch/JitCore_st20c4.c"]),
         Extension("miasm2.jitter.Jitllvm",
                   ["miasm2/jitter/Jitllvm.c"]),
         Extension("miasm2.jitter.Jittcc",
```