Solution du challenge SSTIC 2015

Romain Gayon

12 mai 2015

Résumé

Le challenge SSTIC de cette année consiste à analyser le contenu d'une carte MicroSD insérée dans une clé USB "étrange".

Il nous fera naviguer au travers de moyens exotiques d'administration de système Windows, d'un pilier du jeu vidéo, l'analyse (ou pas) du protocole USB, de traduction de Picsou-JS^{TM}, de reverse de code d'informatique parallèle, pour finir sur une rapide revue de différents formats de fichier image.

Mes moyens de transport favoris seront Ruby et Google.

Table des matières

Introduction

Ce n'est pas la première fois que je m'essaie à la résolution du challenge SSTIC, mais c'est la première fois que l'enchaînement des épreuves m'a motivé à aller jusqu'au bout.

Bien que certains ont, semble-t-il, reproché à cette mouture d'être "trop facile", les étapes du début m'ont permis de m'échauffer, et de me donner confiance. Il est d'autant plus difficile d'abandonner la partie quand on y a déjà investi plusieurs heures et qu'on a relativement simplement gagné des niveaux.

Après ce bref passage Raconte-Ta-Life, un peu de technique.

1 Stage 1

1.1 Une victoire de canard

Le challenge commence lorsque je télécharge le fichier image d'une carte MicroSD à l'adresse : `http://static.sstic.org/challenge2015/challenge.zip`

Une fois décompressée, l'archive `challenge.zip` révèle un fichier `sdcard.img` de 123Mo.

La commande `file` m'apprend qu'il s'agit d'une copie d'une partition VFAT de la carte.

```
$ file sdcard.img
sdcard.img: DOS/MBR boot sector, code offset 0x3c+2, OEM-ID "mkfs.fat",
↪  sectors/cluster 4, root entries 512, Media descriptor 0xf8, sectors/FAT 244,
↪  sectors/track 32, heads 64, sectors 250000 (volumes > 32 MB) , serial number
↪  0xe50d883b, unlabeled, FAT (16 bit)
```

Une fois la partition montée dans un répertoire, je peux en lister le contenu.

```
# mount -o loop sdcard.img root
# ls -lh root
total 33M
-rwxr-xr-x 1 root root 33M mars  26 02:49 inject.bin
```

Ni `file`, ni `strings` ne m'aident à reconnaître ce fichier de 33Mo.

```
$ file root/inject.bin
root/inject.bin: data
$ strings root/inject.bin
$
```

Vu qu'il semble que l'image globale ne contienne pas d'autres données que celles de ce fichier, je tente un `strings` sur `sdcard.img` en retirant ce qui a rapport au système de fichier VFAT.

```
# strings sdcard.img | grep -v -e ".0.1.2.3.4.5.6.7.8.9"
mkfs.fat
NO NAME    FAT16
This is not a bootable disk.  Please insert a bootable floppy and
press any key to try again ...
UILD   SH
zFzF
INJECT  BIN
java -jar encoder.jar -i /tmp/duckyscript.txt
```

J'utilise un premier jeton Google pour me donner des indices. La recherche retourne comme premier résultat :

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

Il s'agit du repository Github du projet "RubberDucky". On se doute maintenant que la clé USB "étrange" est en fait une clé RubberDucky.

Cet appareil qui ressemble à s'y méprendre à une clé USB simule en réalité un clavier qui enverra les touches contenues dans un fichier `input.bin` placé à la racine de la carte MicroSD.

Malheureusement ce fichier est codé dans un format particulier, non interprétable directement.

On utilise un deuxième jeton Google qui me donne en premier résultat un lien vers un fork du repository précédent qui contient un décodeur tout prêt.

```
$ cd USB-Rubber-Ducky/Decode/
$ perl ducky-decode.pl -f /tmp/inject.bin > decoded_inject.bin
```

Le résultat est un fichier texte, d'environ 20000 lignes, qui contient ce que transmet le RubberDucky.

```
$ head -n 20 decoded_inject.bin
 00ff 007d
GUI R

DELAY 500

ENTER

DELAY 1000
 c m d
ENTER

DELAY 50
 p o w e r s h e l l
SPACE
 - e n c
SPACE

 Z g B 1 A G 4 A Y w B 0 A G k A b w B u A C A A d w B y A G k A d A B l A <snip>
```

Le format est facilement compréhensible, des commandes particulières codent des fonctions ou des touches spéciales, et chaque touche à transmettre est séparée par un espace.

Le listing précédent se lit donc ainsi :

1. <Touche Windows>+R
2. Attendre 500ms
3. cmd
4. 3390 fois : lancer une commande `PowerShell -enc <code>`

Lorsqu'il est connecté, le RubberDucky fait ouvrir un invite de commande `cmd.exe`, puis exécute moult commandes `PowerShell` codées.

1.2 Indigestion de Base64

Le manuel de `PowerShell` m'apprend qu'il est capable d'interpréter des commandes qui lui sont passées en Base64 via le switch `-enc`. Reste à décoder le contenu de `decoded_inject.bin`.

Une fois les espace "en trop" supprimés, je vérifie sur la première commande `PowerShell` que le Base64 est bien celui qu'on connaît bien (on ne sait jamais avec Windows).

```
$ dd if=decoded_inject.bin bs=119 skip=1 | head -n 1 | sed 's/ //g' | base64 -d
function write_file_bytes{
        param([Byte[]] $file_bytes, [string] $file_path = ".\stage2.zip");
        $f = [io.file]::OpenWrite($file_path);
        $f.Seek($f.Length,0);
        $f.Write($file_bytes,0,$file_bytes.Length);
        $f.Close();}
function check_correct_environment{
        $e=[Environment]::CurrentDirectory.split("\");
        $e=$e[$e.Length-1]+[Environment]::UserName;
        $e -eq "challenge2015sstic";}
if(check_correct_environment){
        write_file_bytes([Convert]::FromBase64String('UE<snip>UsBzWnXw=='));
        }else{
        write_file_bytes([Convert]::FromBase64String('VA<snip>IAZABlAHIA'));}
```

On obtient bien du "code" `PowerShell` (des retours à la ligne ont été insérés dans le listing ci-dessus pour améliorer la lisibilité), qui prend une chaîne de nouveau codée en Base64 et qui l'écrit dans `stage2.zip`.

Le code se "protège" d'un bête rejeu dans une VM par une vérification sur l'environnement. Il est nécessaire que le répertoire de travail soit `"challenge2015"` et que l'utilisateur soit `"sstic"`.

Le script suivant permet directement d'extraire/décoder `stage2.zip` quand on lui passe `decoded_inject.bin` en argument.

```ruby
#!/usr/bin/ruby
require "base64"
cmd=""
fb3 = File.open("stage2.zip","wb")
File.readlines(ARGV[0]).each do |l|
    case l
    when /- e n c/
        cmd=""
    when /^ ((. )+=) 00a0/
        cmd << $1.scan(/[0-9a-zA-Z=\/]/).join()
        blob1 = Base64.decode64(cmd).delete("\000")
        if blob1=~/FromBase64String\('([^']+)/
            fb3.write(Base64.decode64($1))
        end
    end
end
fb3.close()
puts "stage2.zip decoded"
```

Je décompresse l'archive extraite pour commencer le stage2.

2 Stage 2

L'archive `stage2.zip` contient 3 fichiers :

— `encrypted`, qui contient vraisemblablement les données à déchiffrer ;

— `sstic.pk3`, qui va me rappeler mes plus belles heures de LAN ;

— `memo.txt`, qui contient le message suivant :

```
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
```

Je vérifie que l'empreinte SHA256 du fichier `encrypted` est effectivement celui indiqué dans `memo.txt`. Je connais donc le chiffré, l'algorithme de chiffrement, le vecteur d'initialisation et sa taille, il ne me manque plus que la clé, qui fera donc 256 bits.

2.1 OpenArena, ballade digestive

Le fichier `sstic.pk3` arbore un suffixe bien connu des gamers des temps plus vraiment modernes, celui d'une "map" de Quake3. Il s'agit en réalité d'une archive Zip, qui une fois décompressée révèle une arborescence qui confirme l'hypothèse Quake3.

```
$ unzip -q -d pk3 sstic.pk3
$ find pk3/ -maxdepth 2
pk3/
pk3/AUTHORS
pk3/levelshots
pk3/levelshots/sstic.tga
pk3/README
pk3/maps
pk3/maps/sstic.bsp
pk3/textures
pk3/textures/sstic
pk3/sound
pk3/sound/world
pk3/scripts
pk3/scripts/sstic.arena
```

Le gamer qui sommeille en moi saute sur l'occasion pour installer OpenArena et essayer la map en question, le fichier `pk3/README` a même la gentillesse de me rappeler comment faire.

Je lance le jeu, j'appuie sur la touche [2], et ... rien. Je recommence après avoir changé mon keyboard layout en anglais avec la commande `setxkbmap us`, et je peux enfin charger la carte "`sstic`".

Je suis alors catapulté dans le monde merveilleux et paisible des FPS.

Dès la première salle, je vois apparaître une texture, ou "tile" intéressante :

Elle semble contenir trois fois 4 octets en hexadécimal. S'agirait-il de morceaux de clés ? En poursuivant ma promenade, d'autres textures du même type avec les caractéristiques suivantes apparaissent :

— 4 octets verts ;
— 4 octets blancs ;
— 4 octets oranges ;
— un symbole.

Un rapide parcours de la carte est suffisant pour trouver 4 de ces tiles, moyennant un tir dans un bouton pour révéler une d'entre-elle masquée par une porte coulissante.

Mais quatre tiles ce n'est pas assez. Soit la clé correspond à tous les octets dans une seule couleur, dans ce cas il faudrait trouver 8 tiles, soit les symboles ont une signification et dans ce cas il manque des informations sur leur signification.

Le listing précédent du contenu du fichier `sstic.pk3` était tronqué, le répertoire `pk3/textures/sstic` contient en fait 80 de ces tiles, ainsi que d'autres textures du logo SSTIC et des images de petits démons rigolos.

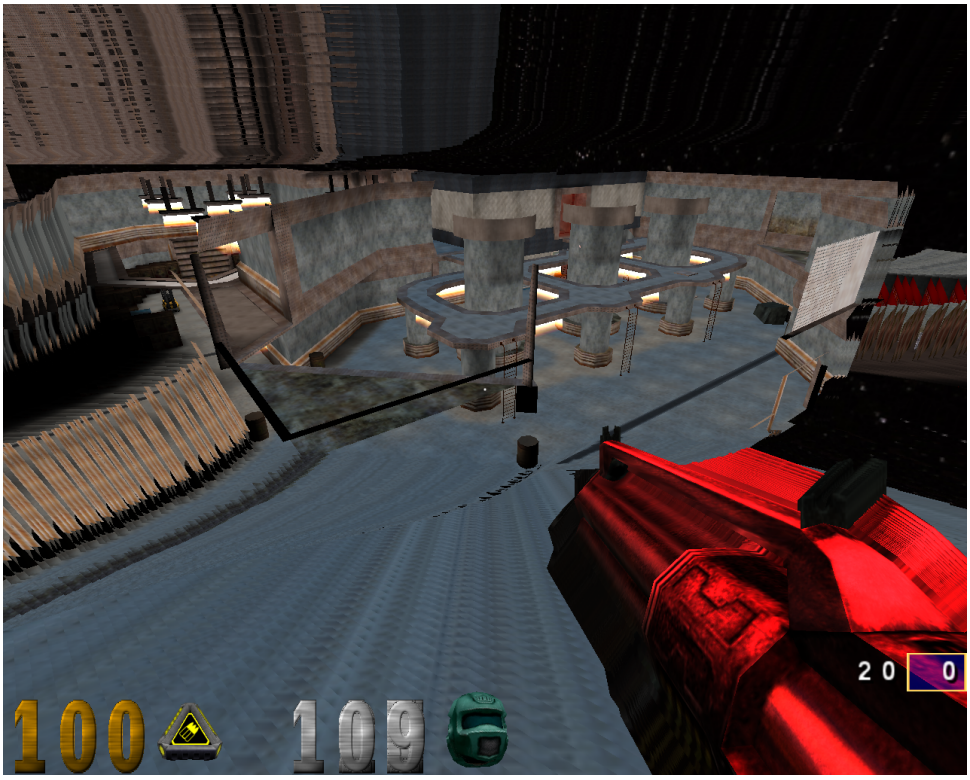
Je pourrais éventuellement réaliser un brute-force de tous les morceaux de clé, mais rien que retranscrire les caractères hexadécimaux, ou faire fonctionner un OCR c'est bien trop fatiguant.

Je me dit que si les tiles sont dans l'archive `pk3`, ils doivent être utilisés quelque part dans la carte. Une première ballade n'en a pas présenté pléthore, celles présentes dans le jeu doivent être effectivement utilisées dans la construction de la clé. Il est temps de se libérer du moteur physique du jeu, devenu gênant.

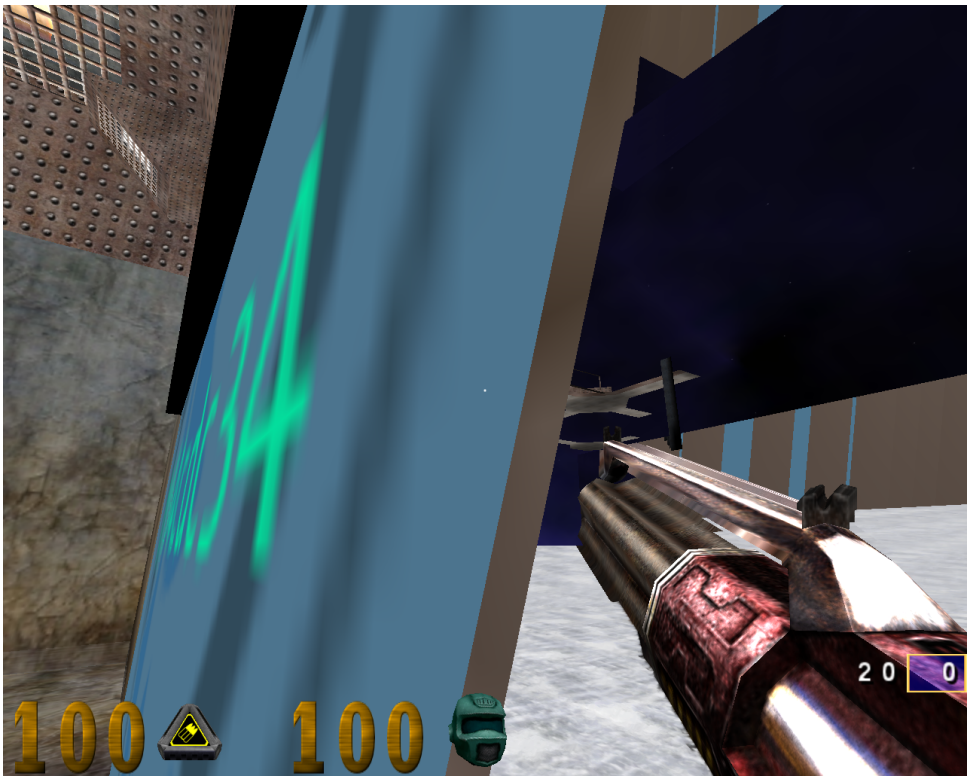
2.2 OpenArena, passage dans la Matrice

Pour la première fois de ce challenge, je vais désobéir aux instructions. De toutes façons, il m'a traité de "noob". Pour charger la map, j'utilise `\devmap sstic` au lieu de `\map sstic`, ce qui permet de charger les commandes "de triche" et notamment `\noclip`.

Cette commande permet de ne plus être soumis aux lois de la gravité du jeu, et également de devenir passe-murailles. Le moteur faisant en sorte de n'afficher une texture que si on est du "bon" côté de celle-ci, je peux avoir une vue aérienne qui pourrait sortir d'un trip aux champignons des prés.



J'explore alors la carte à ma guise et fouille tous les recoins à la recherche des tiles, même derrière le panneau qui masque celle indiquée plus haut.



Sisi, ça passe.

En fouillant la map de fond en comble, aidé du fait que les tiles sur fond bleu se détachent assez bien du décor, j'en découvre 8, chacune avec un symbole différent :

— la tile "sun" sur le mur du début ;
— la tile "blood" sur les caisses de cette mêle salle ;
— la tile "flag" sous une rampe ;

— la tile "wifi" sur un jumper ;
— la tile "pulse" sur un bloc flottant ;
— la tile "link" derrière le panneau coulissant ;
— la tile "pece" dans la cage ;
— la tile "disk" derrière un genre de rocher.

Je retrouve les bonnes tiles et les bonnes séquences d'octet dans le répertoire `pk3/textures/sstic`, que je stocke en attendant dans un fichier `tiles.rb`

```
$sun={  :orange => "626db3b6",
        :white  => "8c34c729",
        :green  => "6043dd54"}
$pece={ :orange => "795fbc7b",
        :white  => "26609fac",
        :green  => "4b763163"}
$wifi={ :orange => "2ce017fd",
        :white  => "30c419d9",
        :green  => "ffe0d355"}
$disk={ :orange => "3d9b0ba6",
        :white  => "d07ccd3d",
        :green  => "ca243465"}
$pulse={:orange => "8267d420",
        :white  => "8153296b",
        :green  => "12f7d028"}
$flag={ :orange => "b0daf152",
        :white  => "db6e3063",
        :green  => "9e2f31f7"}
$link={ :orange => "boob7677",
        :white  => "14e3ec8b",
        :green  => "b54cdc34"}
$blood={:orange => "552f76e6",
        :white  => "7695dc7c",
        :green  => "3acad14b"}
```

Mon exploration au delà des frontières du réel du virtuel, me permet également de découvrir une autre salle, que je n'avais pas vue lors de la ballade.

Je me rapproche un peu plus en nageant dans l'éther.

Cette salle contient la dernière information nécessaire pour reconstituer la clé. Il devient évident que la clé est constituée, pour chaque tile identifiée par son symbole, de la chaîne colorée indiquée.

Je peux alors reconstruire une clé, et la tester en appelant `openssl`. J'ajoute la vérification du SHA256 avec celui de `memo.txt` pour vérifier le script.

```ruby
#!/usr/bin/ruby
$: << File.dirname(__FILE__)
require "digest"
require "tiles.rb"
def test(key)
    iv="535354494332303135 2d537461676532"
    outfile="stage3.zip"
    `openssl aes-256-ofb -d -in encrypted -out #{outfile} -iv #{iv} -K #{key}`
    win="845f8b000f70597cf55720350454f6f3af3420d8d038bb14ce74d6f4ac5b9187"
    puts "WIN" if Digest::SHA256.hexdigest(File.read(outfile)) == win
end

key = [
    $flag[:green],
    $pulse[:white],
    $disk[:orange],
    $blood[:white],
    $flag[:orange],
    $link[:green],
    $wifi[:green],
    $pece[:white]
].join()

test(key)
```

Catelyn ![1] pas de "WIN" à l'horizon. Heureusement le fichier `stage3.zip` se décompresse correctement tout de même. Ouf.

Je peux retrouver le bon hash en supprimant le padding ajouté à l'étape de chiffrement.

```
$ ruby test.rb
$ truncate --size $((`stat -c %s stage3.zip` - 16)) stage3.zip
$ sha256sum stage3.zip
845f8b000f70597cf55720350454f6f3af3420d8d038bb14ce74d6f4ac5b9187  stage3.zip
```

3 Stage 3

L'archive `stage2.zip` contient de nouveau 3 fichiers :
— `encrypted`, très probablement les données à déchiffrer ;
— `paint.cap`, qui devrait contenir un moyen de retrouver la clé ;
— `memo.txt`, qui contient le message suivant :

```
Cipher: Serpent-1-CBC-With-CTS
IV: 0x535354494332303135 2d537461676533
Key: Well, definitely can't remember it... So this time I securely stored it with
↪  Paint.

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

Je dispose comme au stage précédent, de tous les éléments nécessaires pour déchiffrer `encrypted`, sauf la clé.

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

`paint.cap` est une capture de trafic USB, que je m'empresse d'explorer avec `tshark`.

1. Ben oui, "Dame Ned !".

```
$ tshark -r paint.cap
1   0.000000         host -> 3.0          USB 64 GET DESCRIPTOR Request DEVICE
2   0.000613          3.0 -> host         USB 82 GET DESCRIPTOR Response DEVICE
3   0.000666         host -> 2.0          USB 64 GET DESCRIPTOR Request DEVICE
4   0.000850          2.0 -> host         USB 82 GET DESCRIPTOR Response DEVICE
5   0.000884         host -> 1.0          USB 64 GET DESCRIPTOR Request DEVICE
6   0.000891          1.0 -> host         USB 82 GET DESCRIPTOR Response DEVICE
7   2.268500          3.1 -> host         USB 68 URB_INTERRUPT in
8   2.268535         host -> 3.1          USB 64 URB_INTERRUPT in
9   2.284496          3.1 -> host         USB 68 URB_INTERRUPT in
<snip>
```

La capture commence par l'initialisation d'un device USB, puis une communication, au moyen de plus de 28000 "URB"s (pour USB Request Block) entre ce device et l'hôte sur laquelle la capture a été réalisée.

Afin de savoir de quel appareil il s'agit, je m'intéresse à ses réponses lors de la phase d'initialisation.

```
$ tshark -V -r paint.cap frame.number==2 | tail -n 16
DEVICE DESCRIPTOR
bLength: 18
bDescriptorType: 0x01 (DEVICE)
bcdUSB: 0x0200
bDeviceClass: Device (0x00)
bDeviceSubClass: 0
bDeviceProtocol: 0 (Use class code info from Interface Descriptors)
bMaxPacketSize0: 8
idVendor: IBM Corp. (0x04b3)
idProduct: Wheel Mouse (0x310c)
bcdDevice: 0x0200
iManufacturer: 0
iProduct: 2
iSerialNumber: 0
bNumConfigurations: 1
```

`tshark` décode les champs `idVendor` et `idProduct` et montre que l'appareil en question s'annonce comme une classique souris.

La procédure de résolution du stage a l'air relativement évidente. En effet `memo.txt` rappelle que la clé a été "enregistrée" dans le logiciel de dessin Paint, et je dispose de la capture d'une communication USB d'une souris. Il suffira de décoder les mouvements enregistrés par la capture pour découvrir ce qui a été dessiné. Ces mouvements, ainsi que les différents états des boutons de la souris sont facilement décodable des données des trames, comme l'indique le fichier du code source du noyau linux qui s'occupe exactement de cela.

```
$ grep -A 27 "static void usb_mouse_irq" ./drivers/hid/usbhid/usbmouse.c  | tail -n 9
input_report_key(dev, BTN_LEFT,   data[0] & 0x01);
input_report_key(dev, BTN_RIGHT,  data[0] & 0x02);
input_report_key(dev, BTN_MIDDLE, data[0] & 0x04);
input_report_key(dev, BTN_SIDE,   data[0] & 0x08);
input_report_key(dev, BTN_EXTRA,  data[0] & 0x10);

input_report_rel(dev, REL_X,     data[1]);
input_report_rel(dev, REL_Y,     data[2]);
input_report_rel(dev, REL_WHEEL, data[3]);
```

Comme seules les données transmises par la souris ne m'intéressent, ont peut les extraires avec un coup de `tshark`.

```
$ tshark -Y "frame.number >= 7" -r paint.cap -T fields -e usb.capdata | grep ":" >
↪  data.txt
$ head -n 10 data.txt
00:fe:00:00
00:ff:00:00
00:fe:00:00
00:ff:00:00
00:fe:00:00
00:fe:00:00
00:fe:00:00
00:fe:00:00
00:fd:00:00
00:fd:ff:00
```

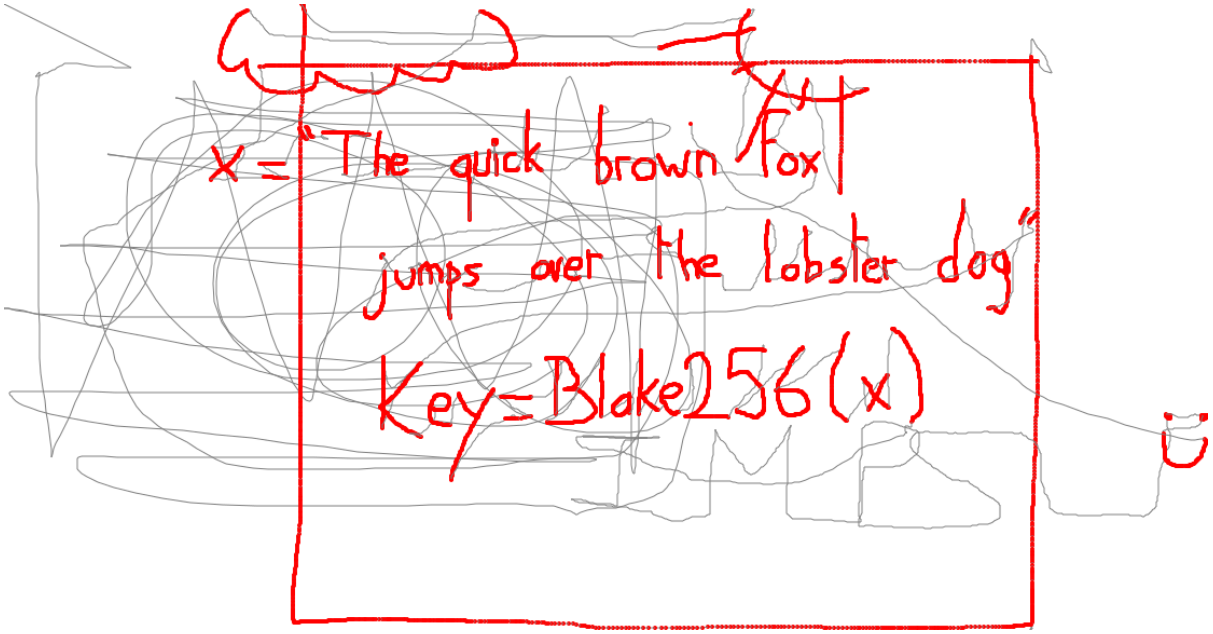
Reste à écrire de quoi décoder les données de chaque trame émise par la souris, et, plus difficile de traduire les mouvements relatifs en dessin, pour retrouver ce qui a été "peint".

Je décide plutôt de demander à Google, tombe dès les premiers résultats sur sur la solution, publiée 3 jours après le lancement du challenge SSTIC, d'un challenge très identique qui s'est déroulé fin février 2015.

Miracle, le code en Ruby est disponible. Il s'occupe d'extraire les coordonnées de mouvements, puis d'utiliser une bibliothèque pour transformer ces mouvements en fichier SVG. Je le copie sans vergogne (vive le libre, qu'on vous dit !), je récupère la bibliothèque `rasem` pour la partie coloriage en SVG, je patche un peu pour que ca fonctionne. Et je lance l'opération de dessin.

```ruby
#!/usr/bin/ruby
$: << "rasem/lib"
require "rasem"

packets = File.readlines('data.txt').collect do |l|
    l.chomp!.gsub!(':', '').scan(/[a-z0-9]{2}/).collect { |b| b.to_i(16) }
end

actions = packets.map do |bytes|
  { :left   => (bytes[0] & (0x01)) != 0,
    :right  => (bytes[0] & (0x02)) != 0,
    :middle => (bytes[0] & (0x04)) != 0,
    :extra  => (bytes[0] & (0x10)) != 0,
    :rel_x      => bytes[1].chr.unpack("c").shift,
    :rel_y      => bytes[2].chr.unpack("c").shift,
    :rel_wheel  => bytes[3].chr.unpack("c").shift
  }
end

poils=[]
x=0
y=0
actions.each do |a|
poil=a
  poil[:x] = x + a[:rel_x]
  x=poil[:x]
  poil[:y] = y + a[:rel_y]
  y=poil[:y]
  poils << poil
end

x_min = poils.map { |a| a[:x] }.min
x_max = poils.map { |a| a[:x] }.max
y_min = poils.map { |a| a[:y] }.min
y_max = poils.map { |a| a[:y] }.max

img = Rasem::SVGImage.new((x_max - x_min),
                          (y_max - y_min)) do
  poils.inject([0, 0]) do |(prev_x, prev_y), action|
    x = action[:x] - x_min
    y = action[:y] - y_min
    line(prev_x, prev_y, x, y, :stroke => '#7f7f7f')
    if action[:left]
      circle(x, y, 2, :fill => 'red')
    end
    [x, y]
  end
end

File.write('mouse.png', img.output)
```

Et je découvre un chef d'œvre de l'art naïf.



En gris, les mouvements de la souris (on notera la signature des petits diablotins malicieux) et en rouge, lorsque le bouton était enfoncé.

En plus d'une magnifique représentation d'un ciel nuageux mais quand même ~~testicule~~ ensoleillé, j'apprends que la clé à utiliser est une empreinte `Blake256` du message `"The quick brown Fox jumps over the lobster dog"`.

Une implémentation de cet algorithme est disponible sur le site de Jean-Philippe Aumasson, et se compile trivialement.

```
$ echo -n "The quick brown Fox jumps over the lobster dog" > key
$ ./blake/blake256 key
4747589afed7e15957a1cbb9efe83436d38d56c63fd8a484317196af81f99524 key
```

Il ne reste plus qu'à trouver une implémentation de l'algorithme de chiffrement `Serpent` et de lui fournir le vecteur d'initialisation de `memo.txt`, le chiffré `encrypted` et la clé trouvée.

Trouver une implémentation "sérieuse" de `Serpent` s'avère plus compliqué qu'il n'y paraît. Cet algorithme, finaliste pour le concours AES, n'a trouvé que peu d'utilisateurs. Toutefois, après de nombreux jetons Google dépensés, je trouve la bibliothèque Crypto++, ainsi que des bindings Ruby adéquats.

Je peux donc tester ma clé avec un nouveau script.

```ruby
#!/usr/bin/ruby
$: << "ruby-cryptopp/ext"
require "cryptopp"
require "digest"

def decrypt(encryped,key,decrypted)
    iv = "53535449433230313352d537461676533"
    serpent = CryptoPP::Serpent.new
    serpent.block_mode = :cbc_cts
    serpent.iv_hex = iv
    serpent.key_hex = key
    File.open(encryped, "rb") do |fi|
        File.open(decrypted, "wb") do |fo|
            serpent.decrypt_io fi, fo
        end
    end

    if Digest::SHA256.hexdigest(File.read(decrypted)) ==
        "7beabe40888fbbf3f8ff8f4ee826bb371c596dd0cebe0796d2dae9f9868dd2d2"
        puts "win"
    end
end

def key(string)
    `echo -n \"#{string}\" > key`
    return `./blake/blake256 key | cut -d " " -f1`
end

k = key(ARGV[0])
puts "key : #{k}"
decrypt("encrypted",k,"stage4.zip")
```

```
$ ruby serpent.rb  "The quick brown Fox jumps over the lobster dog"
key : 4747589afed7e15957a1cbb9efe83436d38d56c63fd8a484317196af81f99524
```

Et, encore une fois, pas de `win`. Pas grave, je me dis que je me suis encore fait avoir par le padding.

```
$ file stage4.zip
stage4.zip: data
```

Tiens non. Je vérifie le texte de l'image dessinée, et je me dit que la casse des caractères n'est pas évidente. Après quelques essais, avec `"The quick brown fox jumps over the lobster dog"` tout rentre dans l'ordre.

```
$ ruby serpent.rb  "The quick brown fox jumps over the lobster dog"
key : 66c1ba5e8ca29a8ab6c105a9be9e75fe0ba07997a839ffeae9700b00b7269c8d
win
$ file stage4.zip
stage4.zip: Zip archive data, at least v2.0 to extract
```

Et je passe au Stage 4.

4 Stage 4

L'archive `stage4.zip` ne contient cette fois-ci qu'un seul document, le fichier `stage4.html`.
Celui-ci se présente comme une page web valide, avec deux blocs distincts :
— une variable `data` de type string, correspondant à environ 516K de caractères hexadécimaux ;
— d'un bloc de code Javascript, sérieusement obfusqué.

Les premières lignes "intéressantes" sont les suivantes :

```
var hash = "08c3be636f7dffd91971f65be4cec3c6d162cb1c";
$=~[];
$={___:++$,$$$$:(![]+"")[$],
__$:++$,$_$_:(![]+"")[$],
_$_:++$,$_$$:({}+"")[$],
$$_$:($[$]+"")[$],
_$$:++$,$$$_:(!""+"")[$],$__:++$,
$_$:++$,$$__:({}+"")[$],$$_:++$,$$$:++$,$___:++$,$__$:++$};
<snip>
```

4.1 Picsou-JS

Le Stage 1 parlait de canard en plastique, ici c'est plutôt l'oncle Picsou qui semble avoir été aux commandes de la moulinette à obfuscation. J'aurais rarement vu autant de symboles `$` à part dans un épisode de La Bande à Picsou (hou hou).

La page s'ouvre parfaitement dans un navigateur Web doté d'un moteur Javascript, par exemple Chromium. Celle-ci retourne un message pas très encourageant.

```
Download manager

Failed to load stage5
```

Je vais alors éclaircir le code en Picsou-JS.

En restant dans le navigateur, j'ouvre les outils de dévelopement Web mis à disposition. Grâce au menu `Developer Tools` et à la console Javascript qui s'affiche, et après avoir rafraichi la page, Chromium présente le code Picsou-JS après une première passe d'interprétation/désobfusction.

Le code Javascript résultant, (cf l'Annexe 8.1) commence à devenir plus lisible. Je distingue l'utilisation de variables faites de "_" et "$", qui seront ensuite remplacées dans le code de chaque fonction aux lignes 66 et suivantes.

Je réalise les remplacements à la main avec un éditeur de textevim y parvient relativement bien, car malheureusement l'opérateur `*`, permettant de sauter d'un token à un autre, ne sait pas s'accomoder de token contenant le caractère $.

Une heure de travail d'artisan du chercher/remplacer et j'ai déjà du code beaucoup plus lisible.

J'ai pu découvrir certaines spécificités du langage JavaScript, comme par exemple la possibilité d'appeler une méthode d'un objet en utilisant la notation "crochet" plutot que "point".

Par exemple : `object.method(var1,var2)` est équivalent à `object['method'](var1,var2)`.

Je supprime la partie du code Picsou-JS chargée d'afficher du code HTML, et il ne reste plus qu'une routine de déchiffrement du contenu de la variable `data`, avec :

— l'algorithme `AES-128-CBC` ;
— le vecteur d'initialisation qui sera la chaîne formée des 16 premier caractères suivant le symbole `'('` du UserAgent du navigateur ;
— la clé sera la chaîne formée des 16 caractères précédant le symbole `')'` du même UserAgent.

On notera que la routine appelle des fonctions de l'interface SubtleCrypto, uniquement implémentées dans des navigateurs récents :

— Firefox $\geq$ 34 ;
— Chrome $\geq$ 37.

Ainsi, pour l'UserAgent `"Mozilla/5.0 (Windows NT 6.3; rv:36.0) Gecko/20100101 Firefox/36.0"`, l'IV sera `"Windows NT 6.3; "` et la clé `" NT 6.3; rv:36.0"`

Après avoir copié le contenu de la variable `data` dans un fichier du même nom, je teste cet UserAgent avec un script. Sans information sur le type de fichier attendu, je suppose pour l'instant qu'il s'agira d'une archive compressée, comme pour tous les stages précédents.

```ruby
#!/usr/bin/ruby
require "openssl"

def aes128_cbc_decrypt(key, data, iv)
    aes = OpenSSL::Cipher.new('AES-128-CBC')
    aes.decrypt ;  aes.key = key ;  aes.iv = iv
    aes.update(data) + aes.final
end

encrypted=File.read("data").scan(/../).map{|l| l.to_i(16).chr}.join()

ua="Mozilla/5.0 (Windows NT 6.3; rv:36.0) Gecko/20100101 Firefox/36.0"
iv=ua[ua.index("(")+1,16]
key=ua[ua.index(")")-16,16]
File.new( "stage5","wb").write( aes128_cbc_decrypt(key,encrypted,iv) )
if `file  stage5 `=~/archive/
    puts "win ! #{ua}"
    exit
end
```

Je lance alors une première fois sans grand espoir.

```
$ ruby aes.rb
aes1.rb:6:in `final': bad decrypt (OpenSSL::Cipher::CipherError)
from aes.rb:6:in `aes128_cbc_decrypt'
from aes.rb:14:in `<main>'
```

4.2 C'est pas moi, m'sieur l'UserAgent

Selon la RFC2616 sur les entêtes d'une requête HTTP, le UserAgent peut être n'importe quelle chaîne de caractères ASCII imprimable, ce qui laisse un nombre beaucoup trop élevé de possibilités pour une attaque naïve par force brute.

Première piste : chercher une liste de plusieurs User-Agent de navigateurs et de les tester un par un.

Des sites comme `http://www.useragentstring.com/` en fournissent plusieurs centaines. Je stocke dans un fichier ceux qui contiennent bien une partie entre parenthèses, et je modifie légèrement mon script précédent.

```ruby
#!/usr/bin/ruby
require "openssl"

def aes128_cbc_decrypt(key, data, iv)
    aes = OpenSSL::Cipher.new('AES-128-CBC')
    aes.decrypt ;  aes.key = key ;  aes.iv = iv
    aes.update(data) + aes.final
end

encrypted=File.read("data").scan(/../).map{|l| l.to_i(16).chr}.join()

ua_tab=File.readlines("uas.lst")


ua_tab.each do |ua|
    iv=ua[ua.index("(")+1,16]
    key=ua[ua.index(")")-16,16]
    next unless iv.size==16
    next unless key.size==16
    begin
        File.new( "stage5","wb").write( aes128_cbc_decrypt(key,encrypted,iv) )
        if `file  stage5 `=~/archive/
            puts "win ! #{ua}"
            exit
        end
    rescue OpenSSL::Cipher::CipherError=> e
        `rm stage5`
    end
end
```

Que je relance, avec un peu plus d'espoir.

```
$ ruby aes2.rb
$
```

Encore perdu. J'envisage alors la piste de la génération automatique d'une liste d'UserAgent valides. Le problème est qu'aucune règle ne définit simplement ce header. Différents éléments tels la version de l'OS, le nom du navigateur, le nom du moteur de rendu et sa version, le nom de différents plugins supplémentaires et leur version, leur ordre, etc. rendent cette piste difficile à exploiter.

On retourne alors par dépit sur le code de `stage4.html`, dont une partie de la page a été ignorée jusqu'à présent.

```
<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>
```

Rien ne paraît utilisable pour me donner des indices sur le User-Agent de la solution. Je tente dans le doute un autre jeton Google pour me renseigner sur ces polices de caractère. Il se trouve que l'article Wikipedia "Lucida Grande" me dit :

> It has been used throughout Mac OS X user interface from 1999 to 2014

Serait-ce un premier indice ? De toute façons, je n'ai pas mieux. Je vais commencer par ne générer que des UserAgent provenant d'un OS supérieur à Mac OS X.

Un deuxième indice provient du code HTML obfusqué dont je me souviens avoir supprimé les lignes.

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

Contrairement à ce que l'on pourrait penser, l'URL ligne 2 est en fait utilisée par les navigateurs Mozilla Firefox

Je vais donc d'abord me restreindre à ne générer que des UserAgent de navigateurs Firefox sous Mac OS X.

Heureusement, Mozilla fournit la méthode par laquelle ces UserAgent sont générés sans que j'ai besoin d'aller lire le code source du navigateur.

J'écris alors une fonction de génération adéquate.

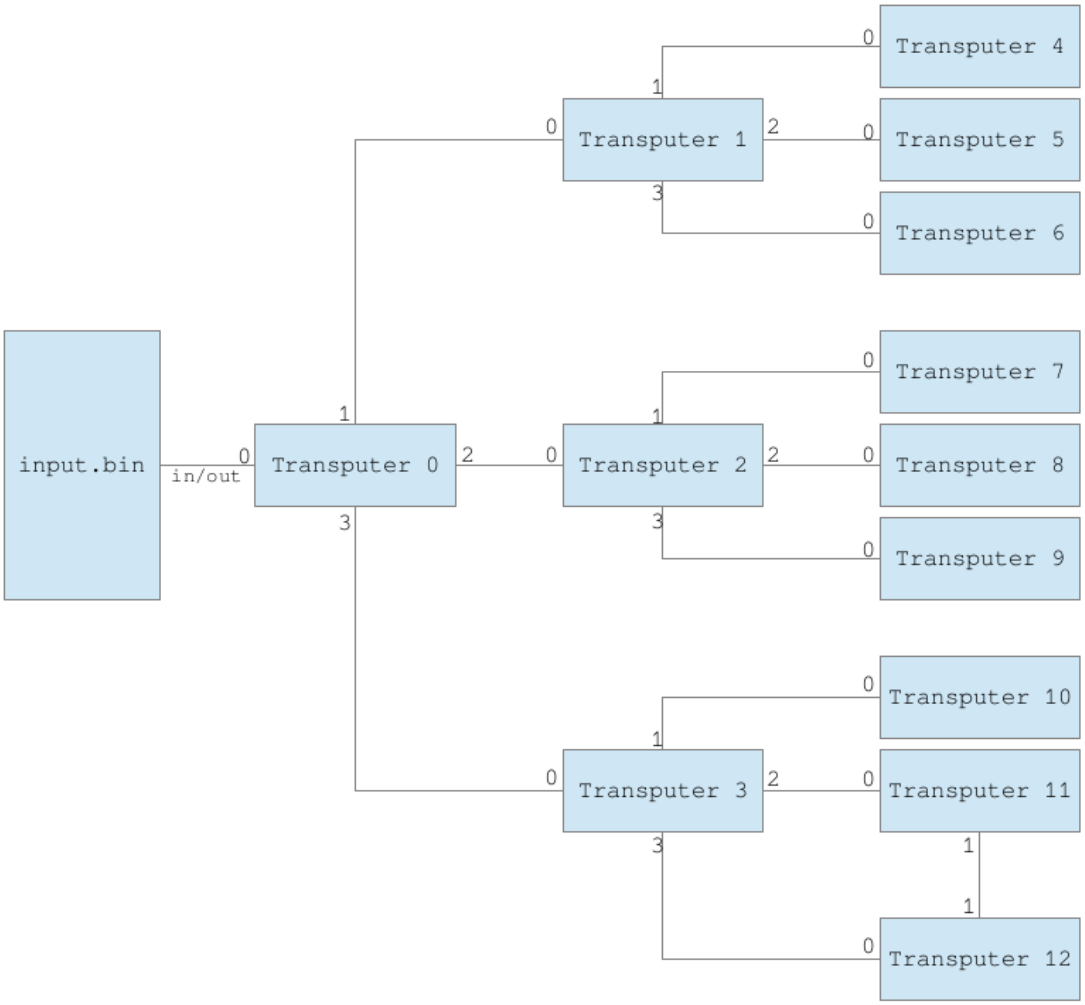
```ruby
#!/usr/bin/ruby
require "openssl"

def aes128_cbc_decrypt(key, data, iv)
    aes = OpenSSL::Cipher.new('AES-128-CBC')
    aes.decrypt ;  aes.key = key ;  aes.iv = iv
    aes.update(data) + aes.final
end

$platforms=
    0.upto(11).map{|i|
        ['Macintosh; Intel Mac OS X','Macintosh; PPC Mac OS X'].map{
            |w| "#{w} 10.#{i}"
        }
}.flatten

$geckoversions=[""].concat(
        0.upto(40).map{|l| 0.upto(20).map{|i| "#{l}.#{i }" } }
).flatten

ua_tab=[]
$platforms.each do |platform|
    ua_tab << "(#{platform})"
    $geckoversions.each do |geckoversion|
        ua_tab << "(#{platform}; rv:#{geckoversion})"
    end
end

encrypted=File.read("data").scan(/../).map{|l| l.to_i(16).chr}.join()

ua_tab.each do |ua|
    iv=ua[ua.index("(")+1,16]
    key=ua[ua.index(")")-16,16]
    next unless iv.size==16
    next unless key.size==16
    begin
        File.new( "stage5","wb").write( aes128_cbc_decrypt(key,encrypted,iv) )
        if `file  stage5 `=~/archive/
            puts "win ! #{ua}"
            exit
        end
    rescue OpenSSL::Cipher::CipherError=> e
        `rm stage5`
    end
end
```

```
$ ruby aes3.rb
win ! (Macintosh; Intel Mac OS X 10.6; rv:35.0)
$ file stage5
stage5: Zip archive data, at least v2.0 to extract
$ mv stage5 stage5.zip
```

Je souffle un peu. Je m'auto-congratule, et surtout je me repose un petit peu avant d'attaquer le stage suivant.

5 Stage 5

L'archive `stage5.zip` contient deux fichiers :
— `input.bin` dont la commande `file` ne connait pas le type ;
— `schematic.pdf` qui me donne deux informations.




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

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

J'ai un schéma et un message me donnant d'abord le SHA256 d'un message chiffré, celui du message déchiffré, puis un ensemble de données de test pour un algorithme inconnu.

5.1 Découverte d'une architecture obsolète

Le schéma relie entre eux des boîtes nommées "Transputers", et le message de test parle d'architecture ST20. Une recherche Google sur chacun de ces deux termes s'impose.

J'apprends alors qu'un Transputer est une plateforme permettant de réaliser des calculs et des algorithmes en parallèle, et qui assure la synchronisation des échanges de données entre chaque nœud.

Le ST20 est une architecture de processeur 32bits, qui a été utilisé notamment sur des plateformes Transputer.

Dans les premiers résultats d'une troisième recherche Google les deux technologies, je trouve un document d'une présentation de cette architecture.

Ce document explique plusieurs points importants pour l'analyse qui va suivre :

— le mode de fonctionnement d'un processeur ST20 (Slide 15 et 16) ;

— l'espace d'adressage (Slide 19) ;

— comment un Transputer peut démarrer à partir de données reçues sur le link0 (Slide 30) ;

J'imagine alors que le dessin du document `schematic.pdf` représente un schéma de connexion de 13 Transputers composés chacun d'un processeur ST20. Il indique également que le fichier `input.bin` est donné en entrée du link0 du premier Transputer.

Comme présenté sur le slide 30 de la présentation de l'architecture, un Transputer peut démarrer sur du code présenté sur son link0. Un premier octet 0xXY lu, supérieur à 0x02, lui indique de lire les 0xXY octets suivants, de les charger à une adresse fixe, puis de placer son pointeur d'instruction `Iptr` à cette adresse pour démarrer l'exécution du code.

```
00000000  F8 64 B4 40 D1 40 D3 24  F2 24 20 50 23 FC 64 B4  .d.@.@.$.$ P#.d.
00000010  2C 49 21 FB 24 F2 48 FB  24 19 24 F2 54 4C F7 24  ,I!.$.H.$.$.TL.$
00000020  79 21 A5 2C 4D 21 FB 24  F2 54 24 79 F7 2C 43 21  y!.,M!.$.T$y.,C!
00000030  FB 24 7A 24 79 FB 61 00  24 19 24 F2 51 4C FB 24  .$z$y.a.$.$.QL.$
00000040  19 24 F2 52 4C FB 24 19  24 F2 53 4C FB 29 44 21  .$.RL.$.$.SL.)D!
00000050  FB 24 F2 48 FB 12 24 F2  54 44 F7 15 24 F2 54 4C  .$.H..$.TD..$.TL
00000060  F7 28 48 21 FB 24 F2 48  FB 13 24 F2 54 41 F7 19  .(H!.$.H..$.TA..
00000070  24 F2 54 13 F1 F7 40 D4  11 24 F2 54 41 F7 15 24  $.T...@..$.TA..$
00000080  F2 51 4C FB 15 24 F2 52  4C FB 15 24 F2 53 4C FB  .QL..$.RL..$.SL.
00000090  10 81 24 F2 55 41 F7 10  82 24 F2 56 41 F7 10 83  ..$.UA...$.VA...
000000A0  24 F2 57 41 F7 10 81 F1  10 82 F1 23 F3 10 83 F1  $.WA.......#....
000000B0  23 F3 10 81 23 FB 11 F1  74 15 F2 F1 74 2C F1 23  #...#...t...t,.#
000000C0  F3 10 23 FB 10 81 F1 74  15 F2 23 FB 74 81 25 FA  ..#....t..#.t.%.
000000D0  D4 CC A3 80 40 D4 10 24  F2 41 FB 66 0B 42 6F 6F  ....@..$.A.f.Boo
000000E0  74 20 6F 6B 00 43 6F 64  65 20 4F 6B 00 44 65 63  t ok.Code Ok.Dec
000000F0  72 79 70 74 00 24 BC 22  F0                       rypt.$.".
```

Reste donc à interpréter les 0xF8+1 octets ci-dessus pour savoir ce que va faire Transputer0. Je nommerai chaque transputerN, "TN" dans le reste du texte.

5.2 Émulateurs

Ceux des autres

Je rentre en plein dans le noyau dur de ce qui compose régulièrement les challenges SSTIC : la rétro-ingénierie

N'ayant environ jamais reversé de code dans ma vie et au lieu d'abandonner à cette étape, je me suis dit qu'utiliser un émulateur serait un bon moyen de continuer à avancer.

Une nouvelle recherche Google fait ressortir le projet ST20EMU qui a la bonne idée de se lancer correctement dans `wine`.

Je crée alors un fichier `boot.bin` contenant le code censé se lancer au tout début, sur T0.

```
$ dd if=input.bin bs=1 skip=1 count=248 > boot.bin
$ wine st20emu.exe
> l 7ff80000 boot.bin
Read 248 bytes from boot.bin
> i 7ff80000
```

Le code de `boot.bin` est chargé à la main à une adresse fixe, et `Iptr` pointe à cette adresse, comme le ferait un Transputer.

Chaque pression sur la touche `Enter` permet d'exécuter instruction par instruction. Je peux voir ainsi des instructions inconnues faire certaines opération sur des registres. Je déchante encore un peu plus lorsque l'instruction `out` doit être exécutée.

```
7ff80016  fb  out
>
This instruction (out) has not been implemented yet
```

Je ne pourrai donc pas utiliser cet émulateur directement.

De nombreux jetons Google sont alors dépensés pour chercher un émulateur complet sans succès. Le code source de `st20emu.exe` est en C pour Windows, pas envie de mettre les doigts dedans.

Qu'à cela ne tienne, il "suffira" de coder l'émulateur de mes rêves moi-même.

Le mien

L'architecture ST20 semble relativement simple. Les opcodes sont sur un ou deux octets, et le ST20 fonctionne avec 6 registres :

- `Optr` qui permet de coder des instructions sur deux octets ;
- `Areg`, `Breg`, `Creg`, trois registres génériques ;
- `Wptr`, un pointeur vers le début de l'espace mémoire de travail ;
- `Iptr`, le pointeur vers l'instruction à exécuter.

Je n'ai plus besoin que de la liste des opcodes pour le processeur ST20, que je trouve assez facilement en demandant gentiment à Google. Je me rends rapidement compte lors du développement de l'émulateur que les opcodes utilisés dans le challenge appartiennent à la version `ST20 C4`. Le document idoine est disponible ici.

Le code de l'émulateur en Ruby est disponible dans l'Annexe 8.2. Voici quelques détails intéressants de l'implémentation :

- chaque Transputer démarre lorsqu'il a lu des données sur son link0 ;
- l'exécution de chaque Transputer est contenue dans un Thread Ruby ;
- la mémoire est implémentée trivialement (ligne 49) par une table de hachage "Adresse mémoire" ⇒ "valeur de l'octet" (ainsi, lire un mot requiert 4 accès à la table de Hash) ;
- la connexion entre deux Transputers est réalisée par un objet Queue, qui permet d'avoir des lecture et des écritures synchronisées (un `.pop()` est bloquant) ;
- il est possible de demander à un objet Transputer d'enregistrer à chaque instruction l'intégralité de son état des registres et de la mémoire (cf ligne 151) ;
- seules les instructions rencontrées lors de l'exécution de `input.bin` ont été implémentées.

Une grosse semaine plus tard, mon émulateur semble fonctionnel. Le processus d'écriture du code fut un peu chaotique mais `st20emu.exe` m'a aidé à rester dans le droit chemin. Je peux enfin émuler `input.bin`.

```
$ ruby transputer.rb input.bin
Boot okCode OkDecrypt?P??"??z!?3^V??!i#??E5?-3???CGw?<snip>
```

Le résultat est encourageant, mais je souhaite pouvoir valider l'émulateur en lui donnant le jeu de test indiqué dans `schematic.pdf`. Comme le fichier `input.bin` est la seule entrée du réseau de Transputers, c'est à l'intérieur de ce fichier qu'il va falloir enregistrer la clé et le texte chiffré du jeu de test.

Pour ce faire, je demande à l'émulateur de tracer l'exécution de T0, et d'afficher les différents appels à l'instruction `in` sur son link0.

Je vois qu'après l'affichage du message `"BootOkCodeOkDecrypt"`, T0 lit 12 octets "FF FF FF FF FF FF FF FF FF FF FF FF" puis lit un octet à la fois à partir de "FE F3 50 DC 81 BC 97 27 89 ...".

Je retrouve facilement ces motifs dans `input.bin`. Les douze FF apparaissent juste après la chaîne `"KEY:"` (offset 0x989 de `input.bin`) et je les remplace avec assez d'assurance par la chaîne `"*SSTIC-2015*"`. Le chiffré de test sera quant à lui inséré à partir de l'offset 0x9AD, qui se trouve juste après la chaîne `"congratulations.tar.bz2"` et j'obtiens le nouveau fichier `test.bin`.

Je relance mon émulateur.

```
$ ruby transputer.rb test.bin
Boot okCode OkDecrypt?I love ST20 architecture
```

J'ai donc un émulateur qui semble fonctionnel. Il me manque "seulement" la clé permettant de déchiffrer le message contenu dans `input.bin`.

Le problème est que l'espace des clés possibles est de $2^{12*8} \approx 7.92*10^{28}$. Pour pouvoir espérer terminer le challenge en moins d'un mois, il faudrait théoriquement pouvoir tester $3 * 10^{22}$ clés par seconde.

Mon émulateur en Ruby est malheureusement légèrement en deçà de ce prérequis.

```
$ time ruby transputer.rb input.bin > congratulations.tar.bz2
real    34m20.986s
```

Cette solution n'est pas valable. Même en supposant réduire de plusieurs exposants le nombre de clés à tester.

5.3 Acculé, on va reverser

Sans indice évident pour trouver la clé à quelques bits près, la seule méthode restante est de reverser le code exécuté par le réseau de Transputers, et de le ré-implémenter. Ainsi, j'espère faire apparaître d'éventuelles faiblesses dans l'algorithme de déchiffrement, et obtenir un code beaucoup plus rapide à exécuter.

Cette étape de reverse tant redoutée va finalement se révéler moins difficile que prévu, pour plusieurs raisons :

— l'écriture de l'émulateur m'a familiarisé avec l'architecture Transputer/ST20 ;
— Google me permet assez facilement de trouver un désassembleur ;
— les parties d'initialisation des 13 Transputers m'est épargnée grâce à l'émulateur ;
— je pourrai valider mes tâtonnements grâce a l'émulateur.

Le code de chaque Transputer est donc repris à la main, et ré-implémenté en Ruby. Parce que le Ruby c'est fun. Parce qu'on peut mettre des `puts toto+"tutu"` partout pour débugger sans se prendre des `SEGFAULT`.

L'émulateur m'a permis de rapidement extraire de `input.bin` les différents extraits de code transmis par T0 aux T[1..3], et de ces derniers aux T[4..12], en s'intéressant particulièrement aux adresses mémoires pointées par `Creg` lors des instructions `out`.

J'ai alors une vue d'ensemble du fonctionnement du réseau de transputers :

— une première phase d'initialisation pendant laquelle T0 transmet successivement les instructions de démarrage à T1, T2, T3
— une seconde phase d'initialisation pendant laquelle T1, T2, T3 chargent chacun les 3 autres Transputers qui leur sont connectés.
— puis pour chaque octet `eb` lu du message chiffré :
  — T0 calcule un XOR `"X"` de chaque octet retourné par T[1..3], calcule $k = xor((index + 2 * key[index]), eb)$
  — T[4..12] retournent chacun un octet généré à partir de la clé.
  — T[1..3] retournent chacun un octet, résultat d'un XOR de chacun des octets retournés par les 3 "sous-transputers"
  — T0 remplace $key[index]$ par X
  — $index = (index + 1) mod (12)$

Le code de `stage5.rb` est disponible dans l'Annexe 8.3.

Sans argument, ce simulateur exécute l'algorithme des Transputers sur le jeu de test.

```
$ ruby stage5.rb
I love ST20 architecture
```

Je peux vérifier que le simulateur est un peu plus rapide que l'émulateur à déchiffrer `encrypted` avec la clé de test.

```
$ time ruby stage5.rb encrypted "428383847367455048495342" > /tmp/test

real    0m4.111s
```

Pour gagner un peu plus de temps, je me motive pour écrire mes premières lignes de C depuis l'école. Heureusement, l'algorithme n'utilise que des calculs assez simple sur des octets, limitant le besoin de faire des opérations sur la mémoire, source de SEGFAULT. De plus, j'ai pu optimiser quelques parcours de tableaux, en commun avec deux Transputers, et regrouper le système décrit dans `schematic.pdf` en une seule grosse fonction (bien dégueulasse).

Le code immonde correspondant est présenté en Annexe 8.4.

```
$ gcc -O3 -DSTANDALONE stage5.c -o stage5
$ time ./stage5

real    0m0.028s
```

Toujours pas assez vite pour pouvoir tester 2^{96} clés.

5.4 Réduction de l'espace des clés possibles

Je suppose que le message déchiffré doit s'appeler `congratulations.tar.bz2`, ce que laisse entendre `input.bin`. Il suivra donc la structure d'un fichier BZip2. Lorsque j'ai remarqué que la clé est utilisée directement pour le calcul des 12 premiers octets du chiffré, je suis allé vérifier à quoi ressemble un entête Bzip2.

Wikipedia indique que l'entête commence toujours, en hexadécimal, `0x425A68`, un octet entre `0x31` et `0x39`, indiquant la taille du bloc de données brutes à compresser à la fois, puis `0x314159265359`. Toutefois, il semble (en vérifiant sur des fichiers BZip2 de mon système), que si le fichier à compresser fait plus de 900ko, la taille de block sera toujours "9", ou `0x39`. Les deux octets suivants dépendent du message que l'on a compressé, dont je ne sais pas la teneur.

Je connais donc 10 des 12 premiers octets du message en clair.

Comme indiqué précedemment, pour les 12 premiers octets, la clé renseignée en entrée est utilisée telle quelle, selon le code de la ligne 109 de `stage5.c`. Je peux donc écrire un script qui calculera tous les octets possibles qui valident cette "équation", pour les 10 premiers octets de clair et le chiffré connus.

```
1  #!/usr/bin/ruby
2
3  encrypted=[0xFE,0xF3,0x50,0xDC,0x81,0xBC,0x97,0x27,0x89,0xAC]
4  decrypted=[0x42,0x5A,0x68,0x39,0x31,0x41,0x59,0x26,0x53,0x59]
5  puts "Clé à tester"
6  0.upto(encrypted.size() -1) do |index|
7      poss=[]
8      0.upto(255) do |b|
9          test = ((index + ((2 * b) & 0xff)) ^ encrypted[index])
10         poss << sprintf("0x%02x",b) if (test & 0xff)==decrypted[index]
11     end
12     puts "char keys#{index}[2] = {"+poss.join(',')+"};"
13 end
```

Ce qui me donne les octets suivants :

```
$ ruby calckey.rb
Clé à tester
char keys0[2] = {0x5e,0xde};
char keys1[2] = {0x54,0xd4};
char keys2[2] = {0x1b,0x9b};
char keys3[2] = {0x71,0xf1};
char keys4[2] = {0x56,0xd6};
char keys5[2] = {0x7c,0xfc};
char keys6[2] = {0x64,0xe4};
char keys7[2] = {0x7d,0xfd};
char keys8[2] = {0x69,0xe9};
char keys9[2] = {0x76,0xf6};
```

Seuls les 2 derniers octets me restent à trouver, je n'ai donc plus que $2^{10} * 256 * 256 \approx 6.7 * 10^7$ clés à tester.

5.5 Le dernier bruteforce

Il ne reste plus qu'à écrire un nouveau code qui itère sur les clés à tester et s'arrête lorsque le message a été correctement déchiffré. Pour cela, je peux tester le SHA256 de chaque tentative et le comparer à

celui indiqué dans `schematic.pdf`. Une autre possibilité, qui évite le calcul d'un SHA256, est de vérifier la présence, en fin du message déchiffré, du footer BZip2 (`0x177245385090`, non aligné sur un octet). Pour ces deux cas, il faut déchiffrer le message en entier afin de savoir si la clé testée est la bonne.

En regardant de plus prêt les premiers octets de fichiers BZip2 présent sur mons système, je réalise que pour un grand nombre d'entre eux, une série de plusieurs 0xFF sont présents. Cette "oracle", s'il est valide, me permettrait de ne déchiffrer que les 32 premiers octets du message chiffré pour chaque test de clé.

Le code correspondant est présent en Annexe 8.5. Il teste chaque clé validant l'"oracle des 4 0xFF" sur le message complet, puis teste le SHA256 de ce dernier.

```
$ make
gcc -O3 -c findkey.c -o findkey.o
gcc -O3 -lcrypto -c stage5.c
gcc -O3 -lcrypto stage5.o findkey.o -o findkey
gcc -O3 -DSTANDALONE stage5.c -o stage5
$ ./findkey
0.75% done, 532568 keys/s
1.49% done, 534492 keys/s
2.24% done, 535246 keys/s
<snip>
39.49% done, 530909 keys/s
Trying key: \x5e\xd4\x9b\x71\x56\xfc\xe4\x7d\xe9\x76\xda\xc5
And the winner is:
\x5e\xd4\x9b\x71\x56\xfc\xe4\x7d\xe9\x76\xda\xc5
Found in 26925766 tries, 49 seconds
```

La clé était donc "`0x5ed49b7156fce47de976dac5`" et je récupère `congratulations.tar.bz2`.

6 Stage 6

6.1 Un dernier petit effort

L'archive `congratulations.tar.bz2` ne contient que le fichier `congratulations.jpg`.

Je comprends que je ne suis pas arrivé au bout de mes peines. La commande `file` m'assure que le fichier est bien un JPG valide, mais je le trouve un peu gros ($\approx$ 250ko) pour l'image qu'il représente. Le JPG est censé être un format de compression après tout.

Je tente de voir si le fichier en contient un autre.

```
$ hachoir-subfile congratulations.jpg
[+] Start search on 252569 bytes (246.6 KB)

[+] File at 0 size=55248 (54.0 KB): JPEG picture
[+] File at 55248: bzip2 archive

[+] End of search -- offset=252569 (246.6 KB)
```

Intéressant. J'extrais ce fichier.

```
$ dd if=congratulations.jpg bs=1 skip=55248 > effort1.tar.bz2
```

6.2 Deux derniers petits efforts

L'archive contient une autre image `congratulations.png`.

Je retente la technique précédente pour voir si un fichier se cache dans celui-ci.

```
$ hachoir-subfile congratulations.png
[+] Start search on 197557 bytes (192.9 KB)

[+] File at 0 size=197557 (192.9 KB): PNG picture: 636x474x32 (alpha layer)

[+] End of search -- offset=197557 (192.9 KB)
```

Échec. Je décide de parser ce fichier PNG avec un petit script Ruby.
Un fichier PNG est composé de différentes parties :
— un header de 8 octets ;
— un ou plusieurs chunks identifié par un type sur 4 caractères, contenant des données.
Pour chaque type de chunk rencontré j'écris une nouvelle branche d'un gros `switch`, puis je rencontre un chunk dont le type, `"sTic"`, n'est pas spécifié dans la norme.
J'extrais les données de ces chunks, dont la commande `file` m'indique qu'il s'agit de `"zlib compressed data"`.

```ruby
require "zlib"

def crc32(lol)
    return Zlib::crc32(lol)
end

s=File.open(ARGV[0],"rb")
color_type=0
header = s.read(8)
zlib = ""
while true do
    length = s.read(4).unpack("N")[0]
    type = s.read(4)
    data = s.read(length)
    crc= s.read(4)
    case type
    when "IHDR"
        lol=data.unpack("NNCCCCC")
        width = lol[0]
        height = lol[1]
        color_type=lol[3]
    when "bKGD"
    when "pHYs"
    when "tIME"
    when "sTic"
        zlib<< data
    when "IDAT"
    when "IEND"
        break
    end
end

out = File.open(ARGV[1],File::CREAT|File::RDWR)
out.write(Zlib::Inflate.inflate(zlib))
out.close()
```

```
$ ruby png.rb congratulations.png out
renzokuken@kirin:~/chall/stage6$ file out
out: bzip2 compressed data, block size = 900k
```

6.3 Trois derniers petits efforts

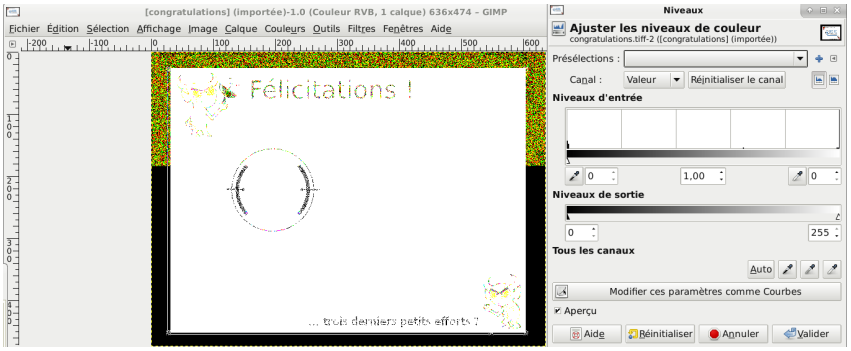
Je décompresse "out", qui contient congratulations.tiff.

`file` m'assure qu'il s'agit d'un fichier TIFF, et `hachoir-subfile` ne trouve pas de fichier trivialement inséré.

Je vais regarder la norme TIFF pour voir si je peux de nouveau écrire un petit parseur, mais le format est un peu plus compliqué que PNG.

J'apprends toutefois que les pixels sont codés en bitmap. Je sais que `congratulations.tiff` n'utilise pas de compression (encore merci à `file`). Je m'attends donc à trouver, en parcourant rapidement le fichier en hexadécimal, de voir les contours noirs, codés `"00 00 00"` puis le reste de l'image. Or il semble que ces pixels ont pour la plupart une composante non nulle.

En jouant avec les niveaux de couleur dans `Gimp` il semble en effet que le haut de l'image contienne de l'information.

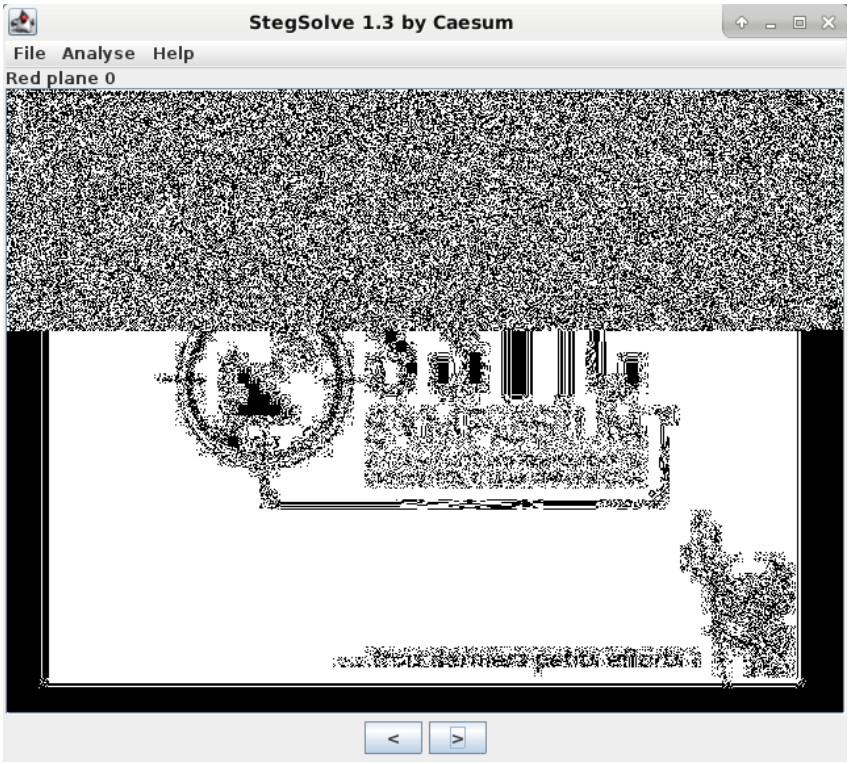


Cette technique, grand classique de la stéganographie, s'appelle "LSB" pour "Least Significant Bit". Il s'agit de coder de l'information sur le dernier bit codant chaque composante de chaque pixel. Ces modifications, invisibles à l'œil nu, apparaissent quand j'augmente artificiellement les différences entre les couleurs, comme sur la capture d'écran précédente.

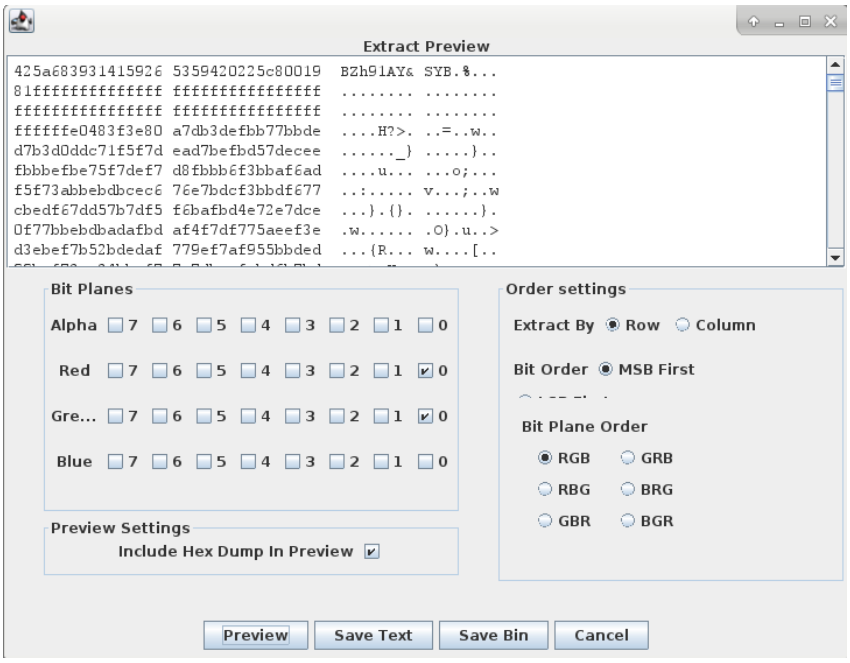
C'est là qu'intervient StegSolve, l'outil classique des challenges et autres CTF pour résoudre ce type de problème.

Je parcours les différents "plans" de l'image pour trouver ceux qui contiennent des données.

Des données se trouvent dans la composante rouge.



Et de la même manière, sur le canal vert. Je me dis que je suis sur la bonne voie quand une "Preview" fait apparaître un entête de fichier maintenant bien connu, que j'extrais directement grâce à l'outil, qui reconstruit chaque bit de données avec 4 pixels (1 bit sur rouge, 1 bit sur vert).



6.4 Quatre derniers petits efforts

`file` m'indique que je suis en présence d'un fichier GIF.

Aucun résultat intéressant avec `hachoir-subfile`. Le fichier semble de taille "normale" par rapport à l'image affichée. Las, ayant toujours `StegSolve` ouvert, je parcours toutes ses options et je vois rapidement que l'adresse email tant rêvée est écrite en bas de l'image, lorsque `Stegsolve` utilise une palette de couleurs aléatoire.

`StegSolve` ne permettant pas de zoomer dans l'image affichée, et les couleurs choisies n'étant pas les plus lisibles, je prends une dernière grande inspiration, et je décide d'écrire un dernier script qui change toutes les couleurs de la palette d'un fichier GIF sauf une. Il ne reste plus qu'à trouver laquelle est le fond du cadre.

```ruby
#!/usr/bin/ruby
# encoding: ascii
io=File.open(ARGV[0],"rb")
res=""
res << io.read(6)
res << io.read(4)
res << io.read(1)
res << io.read(1)
0.upto(255) do |p|
    io.read(3)
    if p == ARGV[1].to_i()
        res << "\x00\xFF\xFF"
    else
        res << "\x00\x00\x00"
    end
end
res << io.read()
print res
```

Et après quelques essais...

```
$ ruby palette.rb  congratulations.gif 2 > lol.gif
```

L'adresse email apparaît "proprement".

J'envoie un mail à l'adresse en question.
Je vais ouvrir une bière.
Libre. (presque).

7 Conclusion

Merci aux concepteurs du challenge d'avoir proposé cette mouture 2015. J'ai apprécié le fait qu'à chaque étape, l'objectif soit assez clair. Je ne me suis jamais senti complètement perdu.

La résolution de certaines étapes a été un peu "mécanique" mais c'est aussi ce qui m'a donné l'impression de progresser dans le challenge et donné envie de continuer.

J'ai également pu confirmer que des bons skills de Google-ing sont nécessaires pour gagner du temps. Il est à mon sens contre-productif d'essayer absolument de réinventer la roue à chaque fois. Il suffit de comprendre ce qu'on fait quand on utilise un outil d'un autre.

Tous les codes que j'ai écrits présentés ici seront disponibles sur mon Github `https://github.com/conchyliculture/sstic2015` après la première semaine de Juin. Peut-être que dans quelques mois, quelqu'un sera heureux d'avoir à sa disposition un demi-émulateur de système à plusieurs Transputers.

8 Annexes

8.1 Picsou-JS -1

```
(function() {
__=document;
$$$='stage5';
$$_$='load';
$_$$=' ';
_$$$$$='user';
_$$$='div';
$$_$$$='navigator';
$$_$$='preferences';
$_$$$='to';
$$$_$='href';
$$$$_='=';
$$$$$='chrome';
_$$$$='"';
$_$$$$='Agent';
$$$_$$='down';
$$$$_$='import';
$='<b>Failed'+$_$$+$_$$$+$_$$+$$_$+$_$$+$$$+'</b>';
___='write';
____='getElementById';
_$_ = "raw";
$$=window;
__$=$$.crypto.subtle;
__$_='decrypt';
___$='status';
$____=$$$$_$+'Key';
_______=0;
__$__='then';
_$____='digest';
__$___='innerHTML';
___$__={name:'SHA-1'};
____$_=data;
_____$=hash;
_$_____=Blob;
___$___=URL;
____$__='createObjectURL';
______$='type';
$_______='application/octet-stream';
_$______='name';
__$_____='AES-CBC';
___$____='iv';
____$___='<a'+$_$$+$$$_$+$$$$_+_$$$$;
_____$__='"'+$_$$+$$$_$$+$$_$+$$$$_+_$$$$+$$$+'.zip'+_$$$$+'>'+$$$_$$+$$_$+$_$$+$$$+'</a>';
______$_='(';
_______$=')';
$________='setTimeout';
_________=parseInt;
__________=$$[$$_$$$][_$$$$$+$_$$$$];
___________='length';
____________='substr';
_____________=_______+1;
______________=_____________*2;
_______________=______________*4;
________________=_______________*______________;
$_$=125*________________;
```

```
__________________='indexOf';
__________________='charCodeAt';
____________________='push';
_______________________=Uint8Array;
________________________='';
_________________________='byteLength';
__________________________=$_$$$+'String';
__[___]('<h'+_______________+'>Down'+$$_$+$_$$+'manager</h'+_______________+'>');
__[___]('<'+_$$$+$_$$+'id'+$$$$_+_$$$$+___$+_$$$$+'><i>'+$$_$+'ing...</i></'+_$$$+'>');
__[___]('<'+_$$$+$_$$+'style'+$$$$_+_$$$$+'display:none'+_$$$$+'><a'+$_$$+'target'+$$$$_+_$$$$+
        'blank'+_$$$$+$_$$+$$$_$+$$$$_+_$$$$+$$$$$+'://browser/content/'+$$_$$+'/'+$$_$$+'.xul'+
        _$$$$+'>Back'+$_$$+$_$$$+$_$$+$$_$$+'</a></'+_$$$+'>');
function _____(______){_=[];
for(__________=_______;
__________<______[___________];
++__________)_[____________________](______[____________________](__________));
return new ______________________(_);
}function ____________________(______){_=[];
for(__________=_______;
__________<______[___________]/_____________;
++__________)_[____________________](_________(______[_____________](
            __________*_____________,_____________),__________________));
return new ______________________(_);
}function ________________________(________){______=________________________;
for(__________=_______;
__________<________[____________________________];
++__________){___=________[__________][____________________________](__________________);
if(___[___________]<_____________)______+=_______;
______+=___;
}return ______;
}function ____________________________(){$_=_____(__________[_____________](
            __________[___________________](______$_)+_____________,__________________));
_$__=_____(__________[_____________](__________[___________________](_______$)-__________________,
            ________________));
_$={};
_$[_$______]=__$_____;
_$[___$____]=$_;
_$[___________]=_$__[___________]*_____________;
__$[$____](_$_,_$__,_$,false,[__$_])[__$__](function(_$___){__$[__$_](_$,_$___,
            ______________________(____$_))[__$__](
            function(___$_){____$ = new ______________________(___$_);
__$[_$____](___$__, ____$)[__$__](function(____$$){if (
            _____$==________________________(new ______________________(____$$))){_____$_={};
_____$_[______$]=$_______;
_____$=new _$_____([____$],_____$_);
__$____ = ___$___[____$__](_____$);
__[____](___$)[__$___]=____$___+__$____+_____$__;
}else{__[____](___$)[__$___]=$;
}});
}).catch(function(){__[____](___$)[__$___]=$;
});
}).catch(function(){__[____](___$)[__$___]=$;
});
}$$[$________](____________________________,$_$);


})
```

8.2 Émulateur

```ruby
#!/usr/bin/ruby

require "pp"
require "thread"

$MOSTNEG    = 0x80000000
$MEMSTART   = 0x80000070
$REGUNDEF   = 0x12345678

$TRUE       = 1
$FALSE      = 0

$stdout.sync=true

# Some debugging help
def word_to_s(w)
    if w
        return sprintf("0x%08x",w)
    else
        return "nil"
    end
end

def w_to_link(w)
    case w
    when 0x80000000
        return "link_0"
    when 0x80000004
        return "link_1"
    when 0x80000008
        return "link_2"
    when 0x8000000C
        return "link_3"
    when 0x80000010
        return "link_0"
    when 0x80000014
        return "link_1"
    when 0x80000018
        return "link_2"
    when 0x8000001C
        return "link_3"
    end
    return "UNKNOWN LINK"
end

# awesome memory implementation:
# just a big Hash
#   address => byte
class MyMemory < Hash
    def read_byte(addr)
        res= self[addr]
        return res || 0xF0
    end

    def read_word(addr,check_align=true)
        if check_align
```

```
            if (addr % 4 !=0)
                raise "Address #{word_to_s(addr)} not aligned"
            end
        end
        val=[]
        val[0]=read_byte(addr) || 0xF0
        val[1]=read_byte(addr+1) || 0xF0
        val[2]=read_byte(addr+2) || 0xF0
        val[3]=read_byte(addr+3) || 0xF0
        # Some mitigation when reading unset memory
        if val.include?(nil)
            if val.uniq() == [nil]
                return nil
            else
                if not check_align
                    return nil
                end
                wtf =  "CHIE : "+word_to_s(addr)+"\n"
                wtf << "CHIE : #{val.join(",")}"+"\n"
                #raise wtf
            end
        end
        res = (val[0]) | (val[1]<<8) | (val[2]<<16) | (val[3]<<24);
        return res
    end

    def write_byte(addr,val)
        self[addr]=val
    end

    def write_word(addr,value)
        val=[]
        val[0] = (value & 0x000000ff)
        val[1] = ((value & 0x0000ff00)>>8)
        val[2] = ((value & 0x00ff0000)>>16)
        val[3] = ((value & 0xff000000)>>24)
        write_byte(addr,val[0])
        write_byte(addr+1,val[1])
        write_byte(addr+2,val[2])
        write_byte(addr+3,val[3])
    end

    def to_s
        res=""
        # Dumps all memory
        self.each_key.sort().each do |k|
            res << sprintf("0x%08x: 0x%02x, ",k,self[k])
        end
        return res +"\n"
    end
end

class Transputer
    attr_accessor :link0,:link1,:link2,:link3,:areg,:breg,:creg,:iptr,:wptr,:num,:debug
    def initialize(num,debug=false)
        @debug=debug
        @areg=$REGUNDEF
        @breg=$REGUNDEF
```

```
115             @creg=$REGUNDEF
116             @oreg=0
117             @wptr=0x80002000
118             @iptr = $MEMSTART
119             @num = num
120             @mem=MyMemory.new()
121             @links_out={}
122             @links_in={}
123             @thread=Thread.new() {
124                 begin
125                     compute()
126                 rescue Exception =>e
127                     raise $!, "Problem with transputer number #{@num}: #{$!}", $!.backtrace
128                 end
129             }
130             @thread.abort_on_exception = true
131             @stack=[] # Stores every state after each execution
132         end
133 
134         def to_s
135             return "Transputer #{@num}"
136         end
137 
138         def kill()
139             @thread.kill()
140         end
141 
142         def set_queue_in(link,q)
143             @links_in[link] = q
144         end
145 
146         def set_queue_out(link,q)
147             @links_out[link] = q
148         end
149 
150         # Display every saved state
151         def dump_stack()
152             res="------------- TRANSPUTER #{@num}\n"
153             @stack.each do |c|
154                 mem=c[:mem]
155                 areginfo = word_to_s(c[:areg])
156                 if c[:areg] and (c[:areg] > 0x60000000)
157                     areginfo << "(" << word_to_s(mem.read_word(c[:areg],false)) << ')'
158                 end
159                 breginfo = word_to_s(c[:breg])
160                 if c[:breg] and (c[:breg] > 0x60000000)
161                     breginfo << "(" << word_to_s(mem.read_word(c[:breg],false)) << ')'
162                 end
163                 creginfo = word_to_s(c[:creg])
164                 if c[:creg] and (c[:creg] > 0x60000000)
165                     creginfo << "(" << word_to_s(mem.read_word(c[:creg],false)) << ')'
166                 end
167 
168                 res << "#{@num} Instr(#{word_to_s(c[:iptr])}) : #{c[:instr]} |"+
169                 " #{c[:instr_txt]}; Areg:#{areginfo}, Breg:#{breginfo}, Creg:#{creginfo}\n"
170                 res << "          Wptr:#{word_to_s(c[:wptr])} oreg:#{word_to_s(c[:oreg])}\n"
171 
172                 res << c[:mem].to_s
```

```
173             if c[:extra]
174                 res << c[:extra]+"\n"
175             end
176         end
177         puts res
178     end
179
180     # Do the thing!
181     def compute()
182         boot_queue = @links_in[:link0]
183         while not boot_queue
184             sleep 0.01
185             boot_queue = @links_in[:link0]
186         end
187         # We only start when data gets available on our link0
188         first = boot_queue.pop()
189         case first
190         when 0
191             raise "in: NOT IMPLEMENTED 0"
192         when 1
193             raise "in: NOT IMPLEMENTED 1"
194         else
195             @code=[]
196 #            puts "TRANSPUTER #{@num} starts and reads #{sprintf("%02x",first)} bytes"
197             # We copy the code we read at address $MEMSTART
198             0.upto(first-1) do |i|
199                 byte = boot_queue.pop()
200                 @mem.write_byte($MEMSTART+i,byte )
201                 @code << byte
202             end
203             if $VERBOSE
204             puts "TRANSPUTER #{@num} starts and has read #{@code.size.to_s(16)} bytes:"+
205                 " #{@code.map{|x| sprintf("%02x",x)}.join()}"
206             end
207             @iptr=$MEMSTART
208         end
209
210         #We can run stuff
211         while l = @mem.read_byte(@iptr);
212             if @debug
213                 stack= {:areg=>@areg,
214                         :breg=>@breg,
215                         :creg=>@creg,
216                         :wptr=>@wptr,
217                         :iptr=>@iptr,
218                         :instr=>"#{l.to_s(16)} (/ #{sprintf("%08x",@iptr)})",
219                         :mem=>@mem.dup
220                 }
221             end
222             instrcode = (l & 0xf0) >> 4
223             instrdata = l & 0x0f
224             @oreg = @oreg | instrdata
225             if @debug
226                 stack[:oreg]=@oreg
227                 @stack << stack unless (instrcode == 0x6 or instrcode == 0x2)
228             end
229             case instrcode
230             when 0x6
```

```
231                 @oreg = (~ @oreg)  << 4
232                 @iptr+=1
233             when 0x2
234                 @oreg= @oreg  << 4
235                 @iptr+=1
236             when 0x0
237                 stack[:instr_txt]= "j #{@oreg.to_s(16)}" if @debug
238                 @iptr+=1
239                 @iptr = @iptr + @oreg
240                 @oreg = 0
241             when 0x1
242                 stack[:instr_txt]= "ldlp #{@oreg.to_s(16)} " if @debug
243                 @creg = @breg
244                 @breg = @areg
245                 @areg = @wptr + (4 * @oreg)
246                 @iptr+=1
247                 @oreg = 0
248             when 0x4
249                 stack[:instr_txt]= "ldc #{@oreg.to_s(16)}" if @debug
250                 @creg = @breg
251                 @breg = @areg
252                 @areg = @oreg
253                 @iptr+=1
254                 @oreg=0
255             when 0x5
256                 stack[:instr_txt]= "ldnlp #{@oreg.to_s(16)}" if @debug
257                 @areg = @areg + (4 * @oreg)
258                 @iptr+=1
259                 @oreg=0
260             when 0x7
261                 stack[:instr_txt]= "ldl #{@oreg.to_s(16)}" if @debug
262                 @creg = @breg
263                 @breg = @areg
264                 @areg = @mem.read_word(@wptr + (4 * @oreg))
265                 stack[:extra] = "       areg res : #{word_to_s(@areg)}" if @debug
266                 @oreg=0
267                 @iptr+=1
268             when 0x8
269                 stack[:instr_txt]= "adc #{@oreg.to_s(16)}" if @debug
270                 @areg=@areg+@oreg
271                 @iptr+=1
272                 @oreg = 0
273             when 0x9
274                 stack[:instr_txt]= "call #{@oreg.to_s(16)}" if @debug
275                 @iptr+=1
276                 @mem.write_word(@wptr - 4, @creg)
277                 @mem.write_word(@wptr - 8, @breg)
278                 @mem.write_word(@wptr - 12, @areg)
279                 @mem.write_word(@wptr - 16, @iptr)
280                 @wptr = @wptr - ( 4 * 4)
281                 @areg = @iptr
282                 @iptr = @iptr + @oreg
283                 @oreg = 0
284             when 0xa
285                 stack[:instr_txt]= "cj #{@oreg.to_s(16)}" if @debug
286                 @iptr+=1
287                 if @areg == 0
288                     @iptr = @iptr + @oreg
```

```
289                 else
290                     @areg = @breg
291                     @breg = @creg
292                 end
293                 @oreg=0
294             when 0xb
295                 stack[:instr_txt]= "ajw #{@oreg.to_s(16)}" if @debug
296                 @wptr= @wptr + (4 * @oreg)
297                 @oreg=0
298                 @iptr+=1
299             when 0xc
300                 stack[:instr_txt]= "eqc #{@oreg.to_s(16)}" if @debug
301                 if @areg == @oreg
302                     @areg = $TRUE
303                 else
304                     @areg = $FALSE
305                 end
306                 @iptr+=1
307                 @oreg=0
308             when 0xd
309                 stack[:instr_txt]= "stl #{@oreg.to_s(16)}" if @debug
310                 @mem.write_word( @wptr+ (4 * @oreg) , @areg)
311                 @areg = @breg
312                 @breg = @creg
313                 @iptr+=1
314                 @oreg=0
315             when 0xe
316                 stnl() # Not implemented
317             when 0xf
318                 case @oreg
319                 when 0x00
320                     stack[:instr_txt]= "rev" if @debug
321                     temp= @areg
322                     @areg = @breg
323                     @breg = temp
324                     @iptr+=1
325                 when 0x01
326                     stack[:instr_txt]= "lb" if @debug
327                     @areg = @mem.read_byte(@areg)
328                     @iptr+=1
329                 when 0x02
330                     stack[:instr_txt]= "bsub" if @debug
331                     @areg = @areg + @breg
332                     @breg = @creg
333                     @iptr+=1
334                 when 0x03
335                     endp() #Not implemented
336                 when 0x04
337                     diff() #Not implemented
338                 when 0x05
339                     add() #Not implemented
340                 when 0x06
341                     stack[:instr_txt]= "gcall" if @debug
342                     temp=@areg
343                     @areg=@iptr
344                     @iptr=temp
345                 when 0x07
346                     q=in_(stack)
```

```
347                 when 0x08
348                     stack[:instr_txt]= "prod" if @debug
349                     @areg = @areg * @breg
350                     @breg = @creg
351                     @iptr+=1
352                 when 0x09
353                     stack[:instr_txt]= "gt" if @debug
354                     if @breg > @areg
355                         @areg = $TRUE
356                     else
357                         @areg = $FALSE
358                     end
359                     @breg = @creg
360                     @iptr+=1
361                 when 0x0a
362                     stack[:instr_txt]= "wsub" if @debug
363                     @areg = @areg + (4 * @breg)
364                     @breg = @creg
365                     @iptr+=1
366                 when 0x0b
367                     out_(stack)
368                 when 0x0c
369                     sub() #Not implemented
370                 when 0x1b
371                     stack[:instr_txt]= "ldpi" if @debug
372                     @iptr+=1
373                     @areg = @iptr + @areg
374                 when 0x1f
375                     stack[:instr_txt]= "rem" if @debug
376                     if @areg == 0
377                         raise Exception.new("WTF REM")
378                     else
379                         @areg = ( @breg % @areg)
380                         @breg = @creg
381                     end
382                     @iptr+=1
383                 when 0x20
384                     stack[:instr_txt]= "ret" if @debug
385                     @iptr = @mem.read_word(@wptr)
386                     @wptr = @wptr + 4 * 4
387                 when 0x33
388                     stack[:instr_txt]= "xor" if @debug
389                     @areg = @breg ^ @areg
390                     @breg = @creg
391                     @iptr+=1
392                 when 0x3b
393                     stack[:instr_txt]= "sb" if @debug
394                     temp = @breg & 0x000000FF
395                     @mem.write_byte(@areg,temp)
396                     @areg = @creg
397                     @iptr+=1
398                 when 0x3c
399                     stack[:instr_txt]= "gajw" if @debug
400                     temp = @areg
401                     @areg = @wptr
402                     @wptr = temp
403                     @iptr+=1
404                 when 0x40
```

```
405                     stack[:instr_txt]= "shr" if @debug
406                     if @areg >= 0 and @areg < 32
407                         @areg = @breg >> @areg
408                     else
409                         puts "WARNING SHR"
410                         @areg= $REGUNDEF
411                     end
412                     @breg = @creg
413                     @iptr+=1
414                 when 0x41
415                     stack[:instr_txt]= "shl" if @debug
416                     if @areg >= 0 and @areg < 32
417                         @areg = @breg << @areg
418                     else
419                         puts "WARNING SHL"
420                         @areg= $REGUNDEF
421                     end
422                     @breg = @creg
423                     @iptr+=1
424                 when 0x42
425                     stack[:instr_txt]= "mint" if @debug
426                     @creg = @breg
427                     @breg = @areg
428                     @areg = $MOSTNEG
429                     @iptr+=1
430                 when 0x46
431                     stack[:instr_txt]= "and_" if @debug
432                     @areg = @breg & @areg
433                     @breg = @creg
434                     @iptr+=1
435                 when 0x5a
436                     stack[:instr_txt]= "dup" if @debug
437                     @creg = @breg
438                     @breg = @areg
439                     @iptr+=1
440                 when 0xc1
441                     stack[:instr_txt]= "ssub" if @debug
442                     @areg = @areg + (2 * @breg )
443                     @breg = @creg
444                     @iptr+=1
445                 else
446                     raise "Unkownk op #{@oreg.to_s(16)}"
447                     exit
448                 end
449                 @oreg =0
450             end # primary opcode
451         end # while true
452     end
453
454     def in_(stack)
455         stack[:instr_txt]= "in => #{w_to_link(@breg)} (#{word_to_s(@areg)}bytes)" if @debug
456         if  @breg >= 0x80000000 and @breg <= 0x8000001C
457             queue = nil
458             case @breg
459             when 0x80000010
460                 queue = @links_in[:link0]
461             when 0x80000014
462                 queue = @links_in[:link1]
```

```
463             when 0x80000018
464                 queue = @links_in[:link2]
465             when 0x8000001C
466                 queue = @links_in[:link3]
467             end
468             if queue
469                 temp=[]
470                 0.upto(@areg-1) do |i|
471                     b = queue.pop()
472                     temp << b
473                     @mem.write_byte(@creg+i,b)
474                 end
475                 stack[:extra]= "Recieved : #{temp.map{|i| sprintf("%02x",i)}.join()}" if @debug
476             else
477                 raise "Warning IN: dest #{@breg} not implemented for other transputers"
478             end
479         else
480             raise sprintf("WARNING in, undefined dest : %08x", @breg)
481         end
482         @iptr+=1
483         return queue
484     end
485
486     def out_(stack)
487         stack[:instr_txt]= "out => #{w_to_link(@breg)}" if @debug
488         queue = nil
489         case @breg
490         when 0x80000000
491             queue = @links_out[:link0]
492         when 0x80000004
493             queue = @links_out[:link1]
494         when 0x80000008
495             queue = @links_out[:link2]
496         when 0x8000000C
497             queue = @links_out[:link3]
498         end
499         if queue
500             count=0
501             0.upto(@areg-1) do |i|
502                 queue.push(@mem.read_byte(@creg+i))
503                 count+=1
504             end
505 #             if @areg==1
506 #                 puts "Transp #{@num} sent #{w_to_link(@breg)} one byte: #{sprintf('%02x',@mem.read
507 #             end
508             stack[:extra]="Transp #{@num} put #{count} bytes for #{w_to_link(@breg)}" if @debug
509         else
510             raise "Transputer#{@num} tried to send #{@areg} data to not connected "+
511                 "#{word_to_s(@breg)}"
512         end
513         if $VERBOSE
514             puts "Transputer #{@num} waiting for #{@areg }output data to be consumed on link"+
515                 " #{word_to_s(@breg)} / #{queue}"
516         end
517         while not queue.empty?
518             sleep(0.001)
519         end
520         @iptr+=1
```

```
521     end
522
523     def outword_(stack)
524         stack[:instr_txt]= "outword " if @debug
525         val=[]
526         val[0] = (@areg & 0x000000ff)
527         val[1] = ((@areg & 0x0000ff00)>>8)
528         val[2] = ((@areg & 0x00ff0000)>>16)
529         val[3] = ((@areg & 0xff000000)>>24)
530         queue = nil
531         case @breg
532         when 0x80000000
533             queue = @links_out[:link0]
534         when 0x80000004
535             queue = @links_out[:link1]
536         when 0x80000008
537             queue = @links_out[:link2]
538         when 0x8000000C
539             queue = @links_out[:link3]
540         end
541         if queue
542             val.each do |i|
543                 queue.push(i)
544             end
545         else
546             raise "Transputer#{@num} tried to send #{@areg} data to not connected"+
547                 " #{word_to_s(@breg)}"
548         end
549         if $VERBOSE
550             puts "Transputer #{@num} waiting for #{@areg }output data to be consumed on link"+
551                 " #{word_to_s(@breg)} / #{queue}"
552         end
553         while not queue.empty?
554         end
555         @iptr+=1
556     end
557 end
558
559 class Transputers
560
561     # This is the "keyboard/monitor" before Transputer0
562     class Dude
563         def initialize()
564             @queue_out=nil
565         end
566
567         def set_queue_in(link,q)
568 #             count=0
569             if link == :link0
570                 t=Thread.new {
571                     temp = ""
572                     while true
573 #                       count+=1
574                         byte = q.pop()
575                         print byte.chr
576 #                       if count == 250629
577                         #    FIn du dechiffrement
578 #                           # Pas de pitié pour les croissants
```

```
#                               exit
#                           end
                      end
                  }
                  t.abort_on_exception = true
              else
                  raise "Fucking link : #{link}"
              end
          end

          def set_queue_out(link,q)
              if link==:link0
                  @queue_out=q
              else
                  raise "Lol"
              end
          end

          def go(io)
              io.each_byte do |b|
                  @queue_out.push(b)
              end
          end
      end

      def initialize
          @all_dem_trans=[]
          @dude = Dude.new()
          @trans0=Transputer.new(0)
          @all_dem_trans << @trans0
          @trans1=Transputer.new(1)
          @all_dem_trans << @trans1
          @trans2=Transputer.new(2)
          @all_dem_trans << @trans2
          @trans3=Transputer.new(3)
          @all_dem_trans << @trans3
          @trans4=Transputer.new(4)
          @all_dem_trans << @trans4
          @trans5=Transputer.new(5)
          @all_dem_trans << @trans5
          @trans6=Transputer.new(6)
          @all_dem_trans << @trans6
          @trans7=Transputer.new(7)
          @all_dem_trans << @trans7
          @trans8=Transputer.new(8)
          @all_dem_trans << @trans8
          @trans9=Transputer.new(9)
          @all_dem_trans << @trans9
          @trans10=Transputer.new(10)
          @all_dem_trans << @trans10
          @trans11=Transputer.new(11)
          @all_dem_trans << @trans11
          @trans12=Transputer.new(12)
          @all_dem_trans << @trans12

          # Now we connect all the transputers together
          set_link(@trans0,:link0,@dude,:link0)
          set_link(@trans0,:link1,@trans1,:link0)
```

```
637             set_link(@trans0,:link2,@trans2,:link0)
638             set_link(@trans0,:link3,@trans3,:link0)
639
640             set_link(@trans1,:link0,@trans0,:link1)
641             set_link(@trans1,:link1,@trans4,:link0)
642             set_link(@trans1,:link2,@trans5,:link0)
643             set_link(@trans1,:link3,@trans6,:link0)
644
645             set_link(@trans2,:link0,@trans0,:link2)
646             set_link(@trans2,:link1,@trans7,:link0)
647             set_link(@trans2,:link2,@trans8,:link0)
648             set_link(@trans2,:link3,@trans9,:link0)
649
650             set_link(@trans3,:link0,@trans0,:link3)
651             set_link(@trans3,:link1,@trans10,:link0)
652             set_link(@trans3,:link2,@trans11,:link0)
653             set_link(@trans3,:link3,@trans12,:link0)
654
655             set_link(@trans4,:link0,@trans1,:link1)
656             set_link(@trans5,:link0,@trans1,:link2)
657             set_link(@trans6,:link0,@trans1,:link3)
658             set_link(@trans7,:link0,@trans2,:link1)
659             set_link(@trans8,:link0,@trans2,:link2)
660             set_link(@trans9,:link0,@trans2,:link3)
661             set_link(@trans10,:link0,@trans3,:link1)
662             set_link(@trans11,:link0,@trans3,:link2)
663             set_link(@trans11,:link1,@trans12,:link1)
664             set_link(@trans12,:link0,@trans3,:link3)
665             set_link(@trans12,:link1,@trans11,:link1)
666
667         end
668
669         def set_link(from_t,from_l,dest_t,dest_l)
670             queue_LR = Queue.new()
671             from_t.set_queue_in(from_l,queue_LR)
672             dest_t.set_queue_out(dest_l,queue_LR)
673             queue_RL = Queue.new()
674             from_t.set_queue_out(from_l,queue_RL)
675             dest_t.set_queue_in(dest_l,queue_RL)
676         end
677
678         def boot(io)
679             @dude.go(io)
680             Thread.new{sleep 1 while true}.join
681         end
682
683         def bailout(e)
684             puts "Bailout!"
685             pp e
686             pp e.backtrace
687             @all_dem_trans.each do |t|
688                 t.kill()
689             end
690             @all_dem_trans.each do |t|
691                 t.dump_stack() if t.debug
692             end
693         end
694     end
```

```
695
696 begin
697     s=Transputers.new()
698     io=File.open(ARGV[0], "rb")
699     s.boot(io)
700 rescue Exception =>e
701     pp e
702     s.bailout(e)
703 end
```

8.3 Simulateur Ruby

```ruby
#!/usr/bin/ruby

$stdout.sync=true

def show_12(l)
    puts    l.map{|b| sprintf("\\x%02x",b) }.join()
end

class Transp0
    def initialize(key)
        @key = key
        if key.size() != 12
            puts "Key needs to be 12 bytes, instead of #{key.size}"
            puts " ex : not '*SSTIC-2015*', but '428383847367455048495342'"
            exit
        end
        @ta = Transp1.new()
        @tb = Transp2.new()
        @tc = Transp3.new()
    end

    def do(string)
        i=0
        res_s=""
        string.each_byte do |eb|
            res=0
            b1 = @ta.do(@key)
            b2 = @tb.do(@key)
            b3 = @tc.do(@key)
            b1 = b3 ^ (b1 ^ b2)
            b0 = (i + 2*@key[i] )  ^eb
            res =b0
            @key[i]=b1 & 0xFF
            res_s << (res  & 0xFF).chr
            $stdout.write (res & 0xFF).chr
            i = (i+1)%12
        end
#         return res_s
    end
end

class Transp1
    def initialize()
        @ta = Transp4.new()
        @tb = Transp5.new()
        @tc = Transp6.new()
    end
    def do(key) # while true
        a = @ta.do(key)
        b = @tb.do(key)
        c = @tc.do(key)
        return (a ^ b) ^ c
    end
end

class Transp2
    def initialize()
```

```
58          @ta = Transp7.new()
59          @tb = Transp8.new()
60          @tc = Transp9.new()
61      end
62      def do(key) # while true
63          a = @ta.do(key)
64          b = @tb.do(key)
65          c = @tc.do(key)
66          return a ^ (b ^ c)
67      end
68  end
69
70  class Transp3
71      def initialize()
72          @ta = Transp10.new()
73          @tb = Transp11_12.new()
74      end
75      def do(key) # while true
76          a = @ta.do(key)
77          b,c  = @tb.do(key)
78          return  a ^ (b ^ c)
79      end
80  end
81
82  class Transp4
83      def initialize
84          @res=0
85      end
86      def do(k)
87          0.upto(11) do |i|
88              @res =( k[i]+@res ) & 0xff # fuck les carry
89          end
90          return @res
91      end
92  end
93
94  class Transp5
95      def initialize
96          @res=0
97      end
98      def do(k)
99          0.upto(11) do |i|
100             @res =( k[i] ^ @res ) & 0xff # fuck les carry
101         end
102         return @res
103     end
104 end
105
106 class Transp6
107     def initialize()
108         @a3 = 0
109         @a1 = 0
110     end
111     def do(k)
112         if  @a3 == 0 # Init
113             0.upto(11) do |i|
114                 @a1 = (@a1+k[i]) & 0xFFFF
115             end
```

```
        end
        @a3 = 1
        shr  = (@a1 & 0x8000) >> 0xF
        shr2 = (@a1 & 0x4000) >> 0xE
        x  = (shr ^ shr2 ) & 0xFFFF
        x2 = (@a1 << 1) & 0xFFFF
        x3 = x ^ x2
        @a1 = x3 & 0xffff
        res = 0xff & x3
        return  res
    end
end

class Transp7
    def do(key)
        a1=0
        a2=0
        0.upto(5) do |i|
            a1 = (a1 + key[i] ) & 0xff
            a2 = a2 + (key[i + 6])
        end
        return (a1 ^ a2) & 0xFF
    end
end

class Transp8
    def initialize()
        @tab=[[0x00]*12] * 4
        @index = 0
    end

    def do(key)
        res=0
        @tab[@index] = key.dup
        @index=(@index + 1) % 4
        0.upto(3) do |j|
            a1=0
            0.upto(11) do |i|
                a1=(@tab[j][i] + a1) & 0xFF
            end
            res = (res ^ a1) & 0xFF
        end
        return res
    end
end

class Transp9
    def do(key)
        a1=0
        0.upto(11) do |i|
            lol= key[i] << ( i & 0x7)
            a1 = (a1 ^ lol  ) &0xff
        end
        return a1
    end
end

class Transp10
```

```
    def initialize()
        @tab = [[0x00]*12]*4
        @index=0
    end

    def do (key)
        res=0
        @tab[@index] = key.dup
        @index = (@index+1) % 4
        @a1 = 0
        0.upto(3) do |i|
            @a1 = ((@tab[i][0] + @a1)  & 0xff)
        end
        lol1 = @tab[(@a1 % 4) ]
        lol2 = (@a1 >> 4)
        res = lol1[(lol2 % 0xc) & 0xff]
        return res
    end
end

class Transp11_12
    def initialize()
        @a3_12 = [0x00]*12
    end
    def do(key)
        res11 = key[(((((@a3_12[1] ^ @a3_12[5]) ^ @a3_12[9]) & 0xff) % 12) & 0xff]
        @a3_12=key.dup

        res12 = key[((((key[0] ^ key[3]) ^ key[7])&0xff) %12) & 0xff]

        return [res11,res12]
    end
end

cle = "*SSTIC-2015*".each_byte.to_a
encrypted="\x1d\x87\xc4\xc4\xe0\xee\x40\x38"+
          "\x3c\x59\x44\x7f\x23\x79\x8d\x9f"+
          "\xef\xe7\x4f\xb8\x24\x80\x76\x6e"
if ARGV.size() == 2
    encrypted=File.read(ARGV[0])
    cle = ARGV[1].scan(/../).map{|i| i.to_i(16)}
end
t0 = Transp0.new(cle)
t0.do(encrypted)
```

8.4 Simulateur C

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>

unsigned int calc_t6(unsigned int in ) {
    unsigned int res=0;
    unsigned int s1=0;
    unsigned int s2=0;
    s1 = (in & 0x8000) >> 0xf ;
    s2 = (in & 0x4000) >> 0xe ;
    res =((s1 ^ s2) & 0xFFFF) ^ ((in << 1) & 0xffff);
    return res;
}

char * decrypt(unsigned char * key, char * encrypted,int enc_len){
    unsigned char cur_key[12] = {0x00};
    unsigned char t1=0;
    unsigned char t2=0;
    unsigned char t3=0;
    unsigned char t4=0;
    unsigned char t5=0;
    unsigned int  t6=0;
    unsigned char t7=0;
    unsigned char t8=0;
    unsigned char t9=0;
    unsigned char t10=0;
    unsigned char t11=0;
    unsigned char t12=0;

    unsigned char t8_a[4][12];
    memset(t8_a, 0, sizeof t8_a);
    int row_8 = 0;

    unsigned char t12_a[12];
    memset(t12_a, 0, sizeof t12_a);

    int i=0;
    int j=0;
    int e=0;

    char * decrypted= malloc ( (sizeof (char) * enc_len) +1);
    decrypted[enc_len] = 0x00;

    int index=0;

    for (i=0;i<12;i++) {
        cur_key[i] = key[i];
    }

    for (i=0;i<12;i++) {
        t6=(t6 + (unsigned int)cur_key[i]) & 0xffff;
    }


    for (e = 0; e < enc_len ; e++) {
        unsigned int t7_1 = 0;
```

```
        unsigned int t7_2 = 0;
        t9=0;
        t8=0;
        index=e%12;

        for (i=0;i<12;i++) {

            t4 = (cur_key[i] + t4) & 0xff;
            t5 = (cur_key[i] ^ t5) & 0xff;
            if (i<6) {
                t7_1 = (t7_1 + cur_key[i]) & 0xff;
                t7_2 = t7_2 + cur_key[i+6];
            }
            t9 = ( t9 ^ ( cur_key[i] << (i & 0x7)  )   ) & 0xff;
        }

        t6 = calc_t6(t6);
        t1 = (t4 ^ t5) ^ (t6 & 0xff);

        t7 = (t7_1 ^ t7_2 ) & 0xff;

        for (i=0;i<12;i++) {
            t8_a[row_8][i] = cur_key[i];
        }
        int lol=0;
        for (j=0;j<4;j++) {
            lol=0;
            for (i=0;i<12;i++) {
                lol = (t8_a[j][i] + lol) & 0xff;
            }
            t8 = (t8 ^ lol) & 0xff;
        }

        t2 = t7 ^ t8 ^ t9;

        unsigned char t10_1=0;
        for (i=0;i<4;i++) {
            t10_1 = ( t8_a[i][0] + t10_1) & 0xff;
        }
        t10 = (t8_a[(t10_1)%4])[ (t10_1 >> 4) % 0xc  ];

        t11 = cur_key[(((((t12_a[1] ^ t12_a[5]) ^ t12_a[9]) & 0xff) % 12) & 0xff];
        for (i=0;i<12;i++) {
            t12_a[i] = cur_key[i];
        }
        t12 = cur_key[(((((cur_key[0] ^ cur_key[3]) ^ cur_key[7])&0xff) %12) & 0xff];

        t3 = t10 ^ t11 ^ t12;

        row_8 = (row_8 + 1) % 4;

        decrypted[e] = (index + ((2 * cur_key[index]) & 0xff)) ^ encrypted[e];
        cur_key[index] = (t3 ^ (t1 ^ t2)) & 0xff ;
    }
    return decrypted;
}

int decrypt_file(char * input, char * output, unsigned char key[12]){
```

```
116     struct stat st;
117     int size;
118     stat(input, &st);
119     size = st.st_size;
120     char *encrypted = malloc(size);
121     char *decrypted = malloc(size);
122     FILE * fi = NULL;
123     FILE * fo = NULL;
124 
125     fi =fopen(input,"r");
126     fread(encrypted, 1, size ,fi);
127 
128     decrypted = decrypt(key,encrypted,size);
129 
130     fo = fopen(output,"w");
131     fwrite(decrypted, size, 1, fo);
132 
133     free(decrypted);
134     free(encrypted);
135     return 0;
136 }
137 
138 void test() {
139     unsigned char key[12] = "\x2a\x53\x53\x54\x49\x43\x2d\x32\x30\x31\x35\x2a";
140     char * encrypted =
141         "\x1d\x87\xc4\xc4\xe0\xee\x40\x38"
142         "\x3c\x59\x44\x7f\x23\x79\x8d\x9f"
143         "\xef\xe7\x4f\xb8\x24\x80\x76\x6e";
144     char * decrypted;
145 
146     decrypted = decrypt(key,encrypted,24);
147     printf("%s\n",decrypted);
148 }
149 
150 int main(int argc, char *argv[])
151 {
152     // SSTIC
153     unsigned char key[12] = "\x2a\x53\x53\x54\x49\x43\x2d\x32\x30\x31\x35\x2a";
154     char * input = "encrypted";
155     char * output = "lol";
156     decrypt_file(input,output,key);
157     return 0;
158 }
```

8.5 Bruteforce C

```c
#include <openssl/sha.h>
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/time.h>
#include "stage5.h"

void * memmem(const void *haystack, size_t haystacklen,
              const void *needle, size_t needlelen);

void show_12(unsigned char key[12]) {
    int i=0;
    for(i=0;i<12;i++) {
        printf("\\x%02x",key[i]);
    }
    printf("\n");
}

/*
 * Vérifie la présence de FF FF FF FF
*/
int check_ffes(char * decrypted, int size) {
    int res =-1;
    void *p;
    char ffes[] = {0xff,0xff, 0xff, 0xff};
    p= memmem(decrypted,size,ffes,sizeof(ffes));
    if (p!=NULL) {
        return 0;
    }
    return res;
}

/*
 * Vérifie le SHA256 du message déchiffré
*/
int check_sha(char * input,int length) {

    unsigned char res[SHA256_DIGEST_LENGTH];
    unsigned char good[SHA256_DIGEST_LENGTH]=
        "\x91\x28\x13\x51\x29\xd2\xbe\x65"
        "\x28\x09\xf5\xa1\xd3\x37\x21\x1a"
        "\xff\xad\x91\xed\x58\x27\x47\x4b"
        "\xf9\xbd\x7e\x28\x5e\xce\xf3\x21";

    SHA256_CTX context;
    SHA256_Init(&context);
    SHA256_Update(&context, (unsigned char*)input, length);
    SHA256_Final(res, &context);

    return memcmp(res,good,SHA256_DIGEST_LENGTH);
}

/*
 * Tente de déchiffrer un message avec une cla
 * Retourne 0 si le SHA256 du déchiffré est le bon
```

```
58 */
59 int try_key(char * filename, unsigned char *key) {
60     int isgood=-1;
61     struct stat st;
62     int size;
63     stat(filename, &st);
64     size = st.st_size;
65     char *encrypted = malloc(size);
66     char *decrypted = malloc(size);
67     FILE * fi = NULL;
68     char output[24]="congratulations.tar.bz2";
69 
70     printf("Trying key: ");
71     show_12(key);
72 
73     fi =fopen(filename,"r");
74     fread(encrypted, 1, size ,fi);
75 
76     decrypted = decrypt(key,encrypted,size);
77 
78     isgood = check_sha(decrypted,size);
79     if (isgood == 0) {
80         printf("And the winner is: \n");
81         show_12(key);
82         decrypt_file(filename,output,key);
83     }
84     free(decrypted);
85     free(encrypted);
86     return isgood;
87 }
88 
89 /*
90  * Test des 1024 * 256 *256 clés différentes
91  * La fonction déchifrera congratulations.tar.bz2 si
92  * la bonne clé est trouvée
93 */
94 void bruteforce_key(char * input) {
95     struct stat st;
96     struct timeval tvs,tve;
97     int size;
98     stat(input, &st);
99     size = 0x32;
100    FILE *fi;
101    long total = 1024 * 256 * 256;
102    char *encrypted = malloc(size);
103    char *decrypted = malloc(size);
104    int a,b,c,d,e,f,g,h,i,j,k,l;
105    int res=-1;
106    int res2=-1;
107    long int tdiff=0;
108    long count;
109    unsigned char *key     = malloc(12);
110    char keys0[2] = {0x5e,0xde};
111    char keys1[2] = {0x54,0xd4};
112    char keys2[2] = {0x1b,0x9b};
113    char keys3[2] = {0x71,0xf1};
114    char keys4[2] = {0x56,0xd6};
115    char keys5[2] = {0x7c,0xfc};
```

```
116     char keys6[2] = {0x64,0xe4};
117     char keys7[2] = {0x7d,0xfd};
118     char keys8[2] = {0x69,0xe9};
119     char keys9[2] = {0x76,0xf6};
120 
121     fi =fopen(input,"r");
122     fread(encrypted, 1, size ,fi);
123 
124     memset(key,0xff,12);
125 
126     gettimeofday(&tvs,NULL);
127 
128     count=0;
129     for (a=0;a<2;a++) {
130     for (b=0;b<2;b++) {
131     for (c=0;c<2;c++) {
132     for (d=0;d<2;d++) {
133     for (e=0;e<2;e++) {
134     for (f=0;f<2;f++) {
135     for (g=0;g<2;g++) {
136     for (h=0;h<2;h++) {
137     for (i=0;i<2;i++) {
138     for (j=0;j<2;j++) {
139     for (k=0;k<256; k++) {
140         for (l=0;l<256; l++) {
141             key[0]  = keys0[a];
142             key[1]  = keys1[b];
143             key[2]  = keys2[c];
144             key[3]  = keys3[d];
145             key[4]  = keys4[e];
146             key[5]  = keys5[f];
147             key[6]  = keys6[g];
148             key[7]  = keys7[h];
149             key[8]  = keys8[i];
150             key[9]  = keys9[j];
151             key[10] = k;
152             key[11] = l;
153             decrypted = decrypt(key,encrypted,24);
154             res = check_ffes(decrypted,24);
155 
156             count+=1;
157             if (count%500000 == 0) {
158                 gettimeofday(&tve,NULL);
159 
160                 tdiff = (tve.tv_usec+1000000*tve.tv_sec) - (tvs.tv_usec+1000000*tvs.tv_sec);
161                 printf("%.2f%% done, %ld keys/s\n",
162                         (float) (100.0 * count)/total,(1000000 * count)/tdiff);
163                 }
164             if (res==0) {
165                 res2 = try_key(input,key);
166                 if (res2 == 0)
167                 {
168                     printf("Found in %ld tries, %ld seconds\n",count,tdiff/1000000);
169                     free(decrypted);
170                     goto end;  // FUCK LE SYSTEM
171                 }
172             }
173             free(decrypted);
```

```
    }}}}}}}}}}}}
end:
    free(encrypted);
}

int main(int argc, char *argv[])
{
    bruteforce_key("encrypted");
    return 0;
}
```

```
$ cat Makefile
FLAGS=-O3

all: findkey stage5

stage5.o: stage5.c
gcc $(FLAGS) -lcrypto -c stage5.c

stage5: stage5.c
gcc $(FLAGS) -DSTANDALONE stage5.c -o stage5

findkey.o: findkey.c
gcc $(FLAGS) -c findkey.c -o findkey.o

findkey: findkey.o stage5.o
gcc $(FLAGS) -lcrypto stage5.o findkey.o -o findkey

clean:
rm *.o stage5 findkey
```