Solution du Challenge SSTIC 2015

Bertrand Danos

06 mai 2015

Table des matières

1 Étape 1 : une clé USB « étrange »

Le challenge SSTIC 2015 consiste à analyser la carte microSD qui était insérée dans une clé USB étrange et à retrouver une adresse email de type @challenge.sstic.org

1.1 Récupération et analyse de l'archive

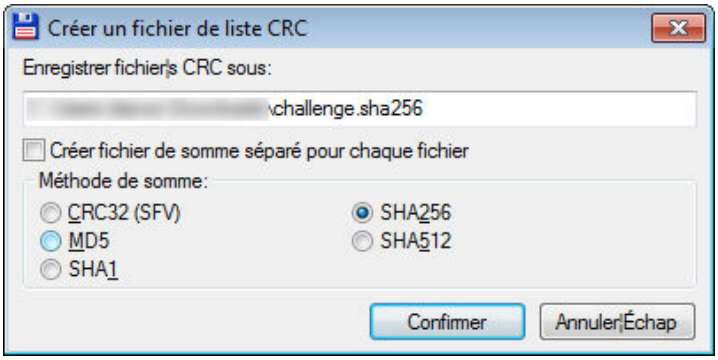
L'épreuve commence par la récupération de l'archive téléchargeable à l'adresse
http://static.sstic.org/challenge2015/challenge.zip

Sous Linux

```
$ wget --quiet http://static.sstic.org/challenge2015/challenge.zip
$ sha256sum challenge.zip
```

Sous Windows

Téléchargement de l'archive via le navigateur
Vérification de la signature avec le gestionnaire de fichiers Total Commander.



Ouverture du fichier challenge.sha256 :
bd0df75a1d6591e01212e6e28848aec94b514e17e4ac26155696a9a76ab2ea31 *challenge.zip

Le hash est le même que celui indiqué sur le site. Le téléchargement s'est donc effectué correctement.

1.2 Découverte du contenu de l'archive

Le fichier téléchargé est ensuite décompressé afin de découvrir son contenu.

Sous Linux

```
$ unzip challenge.zip
```

Sous Windows

Décompression de l'archive avec 7zip

L'archive une fois décompressée a permis d'obtenir un fichier *sdcard.img* de 128Mo.
Ce fichier correspond à une copie à l'identique du contenu de la clé USB.

Les fichiers présents dans cette image sont ensuite extraits.

	**Sous Linux**
	$ sudo mount sdcard.img /media/tmp/
	$ cp /media/tmp/* .
	$ sudo umount /media/tmp

	**Sous Windows** Extraction du contenu du fichier avec 7zip

Un seul fichier est extrait de l'image : inject.bin

Procédons à une analyse rapide du fichier :

	**Sous Linux** $ file inject.bin inject.bin: data

	**Sous Windows** Le gestionnaire de fichier Total Commander, permet de visualiser le contenu d'un fichier en mode hexadécimal.

Cette analyse ne semble fournir aucune indication sur le format du fichier.

Afin d'obtenir plus d'informations, une recherche est effectuée sur Google avec pour mots-clés le nom du fichier "inject.bin".
Google retourne plusieurs informations faisant référence au projet Rubber Ducky :

- http://usbrubberducky.com/?resources#!index.md
- https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Duckyscript

Le fichier inject.bin semble donc être un binaire particulier dédié aux clés USB Rubber Ducky.

1.3 Découverte de la clé USB Rubber Ducky

USB Rubber Ducky est une clé USB qui a la particularité d'être reconnue et de se comporter comme un clavier, sur tous les systèmes d'exploitation. Cette clé est composée d'un processeur 32 bits cadencé à 60Mhz. Cette clé utilise un langage de script, le Ducky Script, exécuté par le processeur pour réaliser une action particulière au clavier.

Pour plus d'informations :

- https://github.com/hak5darren/USB-Rubber-Ducky/wiki
- http://hakshop.myshopify.com/collections/usb-rubber-ducky/products/usb-rubber-ducky-deluxe?variant=353378649
- https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Duckyscript

1.4 Etude du script

Le fichier inject.bin est un script Ducky Script compilé. Pour continuer l'étude du fichier et savoir quelles actions il réalise, il existe 2 solutions :

- Se procurer une clé USB Rubber Ducky.
  Les étapes à réaliser ensuite sont : la copie du fichier inject.bin sur la clé, puis le branchement de la clé sur un ordinateur non connecté au réseau afin de découvrir les actions réalisées.
- Convertir le binaire au format Ducky Script éditable.

La solution consistant à convertir le binaire au format texte Ducky Script est retenue.

Une recherche est effectuée sur Google et permet le téléchargement du script ducky-decode sur https://code.google.com/p/ducky-decode/

Le script téléchargé nécessite d'avoir Perl installé sur le système.

La conversion du fichier inject.bin se réalise ainsi :

Sous Linux et Windows
perl ducky-decode.pl inject.bin > inject.bin_decode.txt

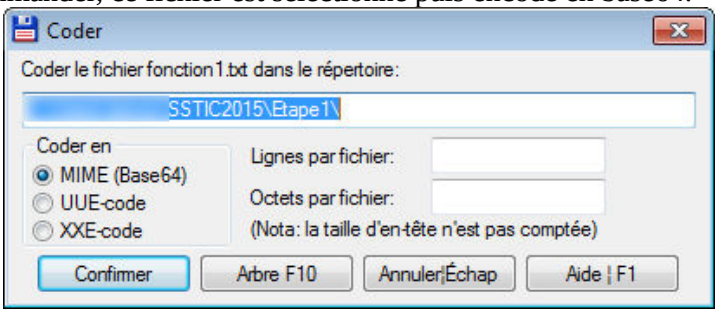
Le fichier *inject.bin_decode.txt* est créé. Il contient toutes les commandes exécutées par la clé.
En ouvrant ce fichier avec un éditeur de texte, on constate qu'il est composé de commandes powershell exécutant des séquences de codes, encodées en base64.

```
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 F 8 A Z g B p A G
ENTER
 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 F 8 A Z g B p A G
ENTER
```

La première séquence en base64 (« ZgB1AG4AYwB ...") est enregistrée dans un fichier command1_b64.txt pour être décodée dans un fichier commande1_decoded.txt

Sous Linux
$ base64 -di command1_b64.txt > commande1_decoded.txt

Sous Windows
Création d'un fichier commande1_decoded.txt
Avec Total Commander, ce fichier est sélectionné puis encodé en base64.



Le fichier créé est édité. Toutes les données encodées sont remplacées par la première ligne de donnée encodée en base64, extraite du fichier *inject.bin_decode.txt*
Total Commander décode ensuite le fichier

Le décodage de la première commande permet d'obtenir un script, encodé en UTF-16.
Ce script est bien entendu toujours écrit en powershell.

```
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
                                                ('contenu en base 64'));
	} else {
		write_file_bytes([Convert]::FromBase64String
                                                ('VAByAHkASABhAHIAZABlAHIA'));
		}
	}
```

L'étude du code, indique qu'un fichier **stage2.zip** est créé. Des données sont écrites dans ce fichier, en se plaçant à la fin de ce fichier. Les données sont donc concaténées par passes successives.
Le script avant de s'exécuter vérifie que son environnement d'exécution est correct : il doit être exécuté depuis un emplacement précis **[X:]\challenge2015** ainsi que par un compte utilisateur particulier : **sstic**. Dans le cas où ces valeurs seraient incorrectes, le script affiche le message "TryHarder" ('*VAByAHkASABhAHIAZABlAHIA*' en base64). Si tout est OK, les données sont écrites

dans le fichier stage2.zip

Chaque ligne powershell du fichier *inject.bin_decode.txt* réalise la même chose. Seules les données à concaténer dans le fichier stage2.zip diffèrent, ce qui est normal.

Le fichier stage2.zip est recrée en réalisant les actions suivantes :

Sous Windows

- Créer un dossier **C:\challenge2015**
- Créer un compte utilisateur **sstic**
- S'identifier avec ce compte
- Regrouper chaque commande powershell dans un fichier batch .cmd.
- Exécuter ce fichier.

Note : si vous ne pouvez créer un compte utilisateur car vous n'êtes pas administrateur de l'ordinateur, vous pouvez utiliser la solution Linux.

Sous Linux

- Placer le script extract_data.py, dont le code est ci-dessous dans le même dossier que le fichier inject.bin_decode.txt
- Exécuter le script :
  $ python extract_data.py
- Le fichier inject_decoded_tmp est obtenu. Ce fichier est encodé en base 64.
- Pour obtenir le fichier stage2.zip, exécuter la commande :

  $ base64 -di inject_decoded_tmp > stage2.zip

```
#!/usr/bin/python

import base64
import codecs

# commandes inutiles
blacklist = ["ENTER\n", "powershell\n", "SPACE\n", "-enc\n", "\n"]

# Suppression des espaces inutiles, ainsi que des commandes inutiles
with open("inject.bin_decode_cleaned.txt", "w") as out:

  linenbr = 1
  with open("inject.bin_decode.txt", "r") as f:
    for line in f:

      # extraction a partir de la ligne 17
      if linenbr<16:
        linenbr += 1
      else:
        cleanedLine = line.replace(" ", "") # Suppresion des espaces
        if cleanedLine: # is not empty
          if cleanedLine not in blacklist:
            out.write(cleanedLine[1160:-233])
```

```
# generation du nouveau fichier
base64.decode(file('inject.bin_decode_cleaned.txt','rb'), file("inject_decoded_tmp",'wb'))
```

Script python extract_data.py

L'analyse du fichier stage2.zip obtenu est faite dans le chapitre suivant.

1.5 Liste des outils utilisés pour résoudre cette épreuve :

	**Logiciels spécifiques Linux** wget sha256sum unzip mount file perl python base64

	**Logiciels spécifiques Windows** Total Commander – http://www.ghisler.com/ 7zip - http://www.7-zip.org/ Perl - https://www.perl.org Powershell

	**Script téléchargé commun aux 2 systèmes** ducky-decode.pl - https://code.google.com/p/ducky-decode/

2 Étape 2 : une partie de Quake pour une énigme

2.1 Découverte du contenu de l'archive

Le fichier téléchargé est ensuite décompressé afin de découvrir son contenu.

Sous Linux
$ unzip stage2.zip

Sous Windows
Décompression de l'archive avec 7zip

Après décompression, le fichier stage2.zip est constitué de 3 fichiers :

- memo.txt : des indications pour résoudre le challenge
- encrypted : le fichier à déchiffrer
- sstic.pk3 : la carte quake3 contenant la clé

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

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

Le fichier memo.txt apporte plusieurs indications :
- le type de chiffrement : AES-OFB.
- le vecteur d'initialisation
- un indice pour trouver la clé : celle-ci est cachée dans la carte quake3
- le hash du fichier chiffré
- le hash du fichier déchiffré

Un aperçu du chiffrement à rétroaction de sortie (Output Feedback - OFB) est expliqué sur wikipedia : http://fr.wikipedia.org/wiki/Mode_d%27op%C3%A9ration_%28cryptographie%29

Vérifions dans un premier temps que l'archive chiffrée est correcte :

Sous Linux

```
$ sha256sum encrypted
91d0a6f55cce427132fc638b6beecf105c2cb0c817a4b7846ddb04e3132ea945  encrypted
```

Sous Windows

```
sha256sum.exe encrypted
91d0a6f55cce427132fc638b6beecf105c2cb0c817a4b7846ddb04e3132ea945  encrypted
```

2.2 Découverte de la carte

Il est nécessaire de disposer de Quake3 ou de sa version libre OpenArena pour pouvoir naviguer dans la carte sstic.pk3.

Dans le cadre du challenge, OpenArena est utilisé.
Voici la procédure pour utiliser la carte au sein du jeu.

Sous Linux

- Installation d'OpenArena :

```
$ sudo apt-get install openarena
```

- Copie de la carte

```
cp sstic.pk3 ~/.openarena/
```

- Démarrage du jeu

```
Démarrer OpenArena, via le menu, ou en ligne de commande
(openarena)
```

- Chargement de la mission

```
Sur l'écran principal, choisir : Multiplayer, puis Create,
puis la carte SSTIC, puis Next, puis Fight
```

La carte SSTIC est une carte sans adversaire. En parcourant la carte, certains éléments du décor sont modifiés et affichent à la place une texture comprenant 3 codes de différentes couleurs ainsi qu'un symbole.
Ces textures sont des indices pour reconstituer la clé de déchiffrement.

Pour accéder à certains lieux, il faut savoir maîtriser le « Rocket Jump », une technique connue des joueurs, ou alors « tricher » en changeant les paramètres de gravité à 0.

Les paramètres du jeu sont modifiables de la manière suivante :
- Activer la console : Appuyer sur shift+Esc
- Mettre la gravité à 0 : Taper '/g_gravity 0' (sans les apostrophes)

Dans la carte se trouve un mur sur lequel tous les symboles sont présents ainsi que le code couleur à utiliser :

La clé correspond donc au code couleur correspondant au symbole, soit :

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

2.3 Déchiffrage du fichier

Le script suivant est à exécuter avec python3 pour déchiffrer l'archive :

```
from Crypto.Cipher import AES

IV = bytes.fromhex('535354494332303135 2d537461676532')
key =
bytes.fromhex('9e2f31f78153296b3d9b0ba67695dc7cb0daf152b54cdc34ffe0d35526609fac')

decryptor = AES.new(key, AES.MODE_OFB, IV)
with open('decrypted', 'wb') as decrypted:
        with open('encrypted', 'rb') as encrypted:
                while True:
                        chunk = encrypted.read(1024)
```

```
            if len(chunk) == 0:
                break
            decrypted.write(decryptor.decrypt(chunk))
```

2.4 Liste des outils utilisés pour résoudre cette épreuve :

	**Logiciels spécifiques Linux** unzip sha256sum OpenArena Python3

	**Logiciels spécifiques Windows** 7zip - http://www.7-zip.org/ Sha256sum - http://www.labtestproject.com/files/win/sha256sum/sha256sum.exe OpenArena - http://openarena.ws/ Python3 - https://www.python.org/

3 Étape 3 : pêche USB : chien homard

3.1 Découverte du contenu de l'archive

Le fichier déchiffré lors de l'étape précédente est décompressé afin de découvrir son contenu.

Sous Linux
$ unzip decrypted.zip

Sous Windows
Décompression de l'archive avec 7zip

Après décompression, le fichier decrypted.zip est constitué de 3 fichiers :

- memo.txt : des indications pour résoudre le challenge
- encrypted : le fichier à déchiffrer
- paint.cap : un fichier de capture de trames au format Wireshark

```
Cipher: Serpent-1-CBC-With-CTS
IV: 0x53535449433230313352d537461676533
Key: Well, definitely can't remember it... So this time I securely stored it with Paint.

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

Le fichier memo.txt apporte plusieurs indications :
- le type de chiffrement : Serpent-1
- le vecteur d'initialisation
- un indice pour trouver la clé : celle-ci a été dessinée sous Paint
- le hash du fichier chiffré
- le hash du fichier déchiffré

3.2 Découverte du fichier de capture

Le fichier à analyser est une capture de flux USB réalisée avec Wireshark.

```
Frame 2: 82 bytes on wire (656 bits), 82 bytes captured (656 bits)
USB URB
DEVICE DESCRIPTOR
   bLength: 18
   bDescriptorType: DEVICE (1)
   bcdUSB: 0x0200
   bDeviceClass: Device (0x00)
   bDeviceSubClass: 0
   bDeviceProtocol: 0 (Use class code info from Interface Descriptors)
```

```
bMaxPacketSize0: 8
idVendor: IBM Corp. (0x04b3)
idProduct: Wheel Mouse (0x310c)
bcdDevice: 0x0200
iManufacturer: 0
iProduct: 2
iSerialNumber: 0
bNumConfigurations: 1
```

Le périphérique capturé est une souris.
Le fichier contient donc tous les mouvements de la souris ainsi que les boutons pressés.

L'objectif de cette épreuve est donc de faire un rejeu des mouvements de la souris et de les redessiner pour obtenir le mot de passe.

3.3 Protocole souris

Le protocole souris est composé de 4 octets, dont voici la représentation :

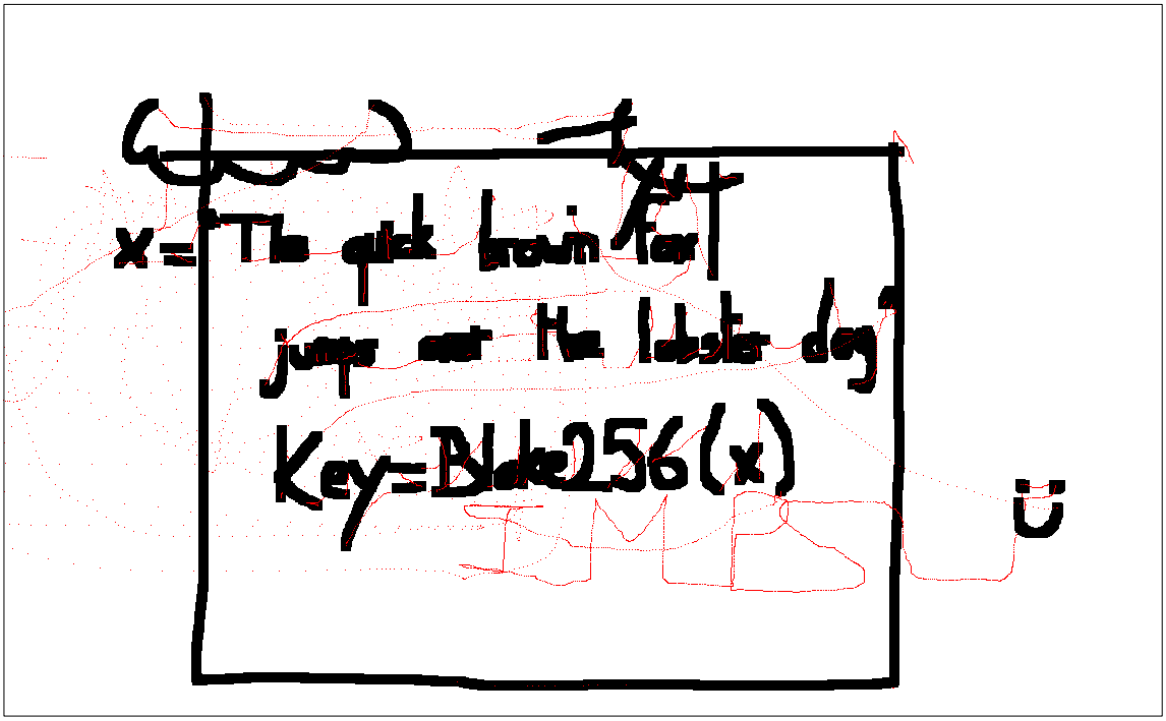
	7	6	5	4	3 2	1	0
Octet 0	0	0	0	0	0 0	Click droit	Click gauche
Octet 1	Déplacement sur l'axe X (signé)						
Octet 2	Déplacement sur l'axe Y (signé)						
Octet 3	Déplacement sur l'axe Z (signé)						

3.4 Rejeu des mouvements de la souris

En recherchant sur Google, un script réponds aux besoins : il permet d'extraire les positions de la souris ainsi que les click pour ensuite en dessiner une image.
http://wiremask.eu/boston-key-party-2015-riverside/

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

import struct
import Image
import dpkt

INIT_X, INIT_Y = 100, 400

def print_map(pcap, device):
  picture = Image.new('RGB', (1200, 1000), 'white')
  pixels = picture.load()
  counter = 0

  x, y = INIT_X, INIT_Y
```

```
  for ts, buf in pcap:
    device_id, = struct.unpack('b', buf[0x0B])

    if device_id != device:
      continue

    data = struct.unpack('bbbb', buf[-4:])

    status = data[0]
    x = x + data[1]
    y = y + data[2]
    print("packet:%d, x=%d, y=%d") % (counter, x, y)

    if (status == 1):
      for i in range(-5, 5):
        for j in range(-5, 5):
          pixels[x + i , y + j] = (0, 0, 0, 0)
    else:
      try:
        if x<0:
          print "valeur evitee"
        else:
          pixels[x, y] = (255, 0, 0, 0)
      except Exception as e:
        picture.save('riverside_map.png', 'PNG')
          raise
    counter+=1

  picture.save('paint.png', 'PNG')

if __name__ == '__main__':

  f = open('paint.cap', 'rb')
  pcap = dpkt.pcap.Reader(f)

  print_map(pcap, 3)
  f.close()
```



On peut lire sur l'image :

x= "The quick brown fox jumps over the lobster dog"
Key=Blake256(x)
IMPS

3.5 Déchiffrage du fichier

Blake256 est une fonction de hashage
(http://en.wikipedia.org/wiki/BLAKE_%28hash_function%29)
Il existe une implémentation python de cette fonction, disponible ici :
http://www.seanet.com/~bugbee/crypto/blake/blake.py

Le script suivant permet d'obtenir la clé :

```
from binascii import hexlify
from blake import BLAKE

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

La clé est la suivante :
66c1ba5e8ca29a8ab6c105a9be9e75fe0ba07997a839ffeae9700b00b7269c8d

Le fichier encrypted est déchiffré en utilisant le site http://serpent.online-domain-tools.com/
Le site fournit un fichier : odt-IV-535354494332303135 2d537461676533.dat

L'analyse du fichier permet d'en détermine le type

	**Sous Linux** $ file odt-IV-535354494332303135 2d537461676533.dat odt-IV-535354494332303135 2d537461676533.dat: Zip archive data, at least v2.0 to extract

Pour ouvrir ce fichier, nous utiliserons la commande jar

	**Sous Linux** $ jar xvf odt-IV-535354494332303135 2d537461676533.dat décompressée: stage4.html

Nous obtenons le fichier stage4.html

3.6 Liste des outils utilisés pour résoudre cette épreuve :

	**Logiciels spécifiques Linux** unzip Python Wireshark jar

	**Logiciels spécifiques Windows** 7zip - http://www.7-zip.org/ Python - https://www.python.org/ Wireshark -

4 Étape 4 : web crypto

4.1 Découverte de la page HTML

Le fichier stage4.html est une page web constituée de 3 éléments stockés à l'intérieur de balises <script>:
- une variable data qui contient des données encodées en hexa
- un hash
- un code javascript obfusqué.

```
<html>
<head>
<style>
	* { font-family: Lucida Grande,Lucida Sans Unicode,Lucida Sans,Geneva,Verdana,sans-serif; text-align:center; }
	#status { font-size: 16px; margin: 20px; }
	#status a { color: green; }
	#status b { color: red; }
</style>
</head>
<body>
	<script>
		var data = "2b1f25cf8db5d243f59b065da6b56……."
		var hash = "08c3be636f7dffd91971f65be4cec3c6d162cb1c";
		$=~[];$={___:++$,$$$$:(![]+"")[$],__$:++$,$_$_:(![]+"")[$],_$_:++$,$_$$: ….
	</script>
</body>
</html>
```

L'aperçu de la page html est le suivant :

Download manager

Failed to load stage5

4.2 Identification de l'outil d'obfuscation

Le code JavaScript possède la particularité de ne contenir aucun nom de variable. Celles-ci ont été remplacées par des caractères $ et _.

En faisant une recherche sur google avec les premiers caractères du code javascript « $=~[] », le lien suivant est proposé: https://blog.korelogic.com/blog/2015/01/12/javascript_deobfuscation
Il permet de découvrir que l'outil utilisé pour obfusquer le code javascript s'appelle JJEncode et qu'il

existe des solutions pour retrouver le code d'origine.

4.3 Désobfuscation du code

L'outil python-jjdecoder est utilisé pour obtenir un code plus compréhensible.
En éditant le code et en faisant des substitutions manuelles le code entier est finalement reconstitué.

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


function getUint8Array(dataAndEvents) {
  array = [];
  for (i=0;i < dataAndEvents.length;++i) {
    array.push(dataAndEvents.charCodeAt(i));
  }
  return new Uint8Array(array);
}

function getUint8ArrayParseInt(dataAndEvents) {
  array = [];
  for (i=0;i < dataAndEvents.length / 2;++i) {
    array.push(parseInt(dataAndEvents.substr(i * 2, 2), 16));
  }
  return new Uint8Array(array);
}

function getStr(deepDataAndEvents) {
  str = "";

  for (i=0;i < deepDataAndEvents.byteLength;++i) {
    write = deepDataAndEvents[i].toString(16);
    if (write.length < 2) {
      str += 0;
    }
    str += write;
  }
  return str;
}

function decrypt_data_and_hash_check() {
  iv =
getUint8Array(window.navigator.userAgent.substr(window.navigator.userAgent.indexOf("(") + 1,
16));
  key =
getUint8Array(window.navigator.userAgent.substr(window.navigator.userAgent.indexOf(")") - 16,
16));

  crypto_parameters = {};
```

```
  crypto_parameters["name"] = "AES-CBC";
  crypto_parameters["iv"] = iv;

  /** @type {number} */
  crypto_parameters["length"] = key["length"] * 8;

          // usage : window.crypto.subtle.importKey("jwk", jwkKey, "RSAES-PKCS1-v1_5", "true",
["encrypt", "verify"]).then(handleImport);
  window.crypto.subtle.importKey("raw", key, crypto_parameters, false, ["decrypt"]).
  then(function(deepDataAndEvents) {

    // usage : window.crypto.subtle.decrypt("RSAES-PKCS1-v1_5", prvKeyEncrypt, pMessage);
          window.crypto.subtle.decrypt(crypto_parameters, deepDataAndEvents,
getUint8ArrayParseInt(data)).
          then(function(dataAndEvents) {

      decrypted_data = new Uint8Array(dataAndEvents);
      window.crypto.subtle.digest({name : "SHA-1"}, decrypted_data).
            then(function(dataAndEvents) {

        if (hash == getStr(new Uint8Array(dataAndEvents))) {

          blob_options = {};
          blob_options["type"] = "application/octet-stream";

          hash = new Blob([decrypted_data], blob_options);

          url = URL.createObjectURL(hash);
          document.getElementById("status").innerHTML = '<a href="' + url +
'download="stage5.zip">download stage5</a>';
        } else {
          document.getElementById("status").innerHTML = "<b>Failed to load stage5</b>";
        }
      });
    }).catch(function() {
      document.getElementById("status").innerHTML = "<b>Failed to load stage5</b>";
    });
  }).catch(function() {
    document.getElementById("status").innerHTML = "<b>Failed to load stage5</b>";
  });
}

window.setTimeout(decrypt_data_and_hash_check, 1000);
```

4.4 Etude du code javascript

Le script possède un lien caché vers les préférences de Firefox.

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

```
preferences</a></div>');
```

Le code utilise la Web Cryptography API (utilisation des fonctions window.crypto.subtle.*)
Ces fonctions sont disponibles actuellement uniquement pour les navigateurs suivants :

- Chrome >v37
- Firefox >v34
- Safari >v8

Source : https://developer.mozilla.org/fr/docs/Web/API/SubtleCrypto

Les informations suivantes sont également extraites :

```
 iv =
getUint8Array(window.navigator.userAgent.substr(window.navigator.userAgent.indexOf("(") + 1,
16));
 key =
getUint8Array(window.navigator.userAgent.substr(window.navigator.userAgent.indexOf(")") - 16,
16));

 crypto_parameters = {};
 crypto_parameters["name"] = "AES-CBC";
 crypto_parameters["iv"] = iv;
```

Avec ces informations, nous avons l'algorithme, l'IV et la clé pour déchiffrer la variable data.
L'IV et la clé sont obtenus à partir du useragent du navigateur. Le chiffrement est AES-CBC.

Le navigateur qui dispose du bon useragent peut donc déchiffrer l'archive.

4.5 Brute-Force du User-Agent

Avec l'aide de certains sites spécialisés, une base de donnée des useragent a été constituée.
Au moment du test, la base est constituée de plus de 830.000 valeurs.
Tester chaque valeurs prendrait trop de temps. Seules les valeurs les plus pertinentes seront extraites de la base : à savoir les navigateurs listés ci-dessus qui supportent la web crypto api.
De plus, le lien « chrome:// » permet de restreindre les valeurs au seul navigateur firefox >v34.

Une analyse en bruteforce avec javascript plante le navigateur assez rapidement.
Il est donc préférable de déchiffrer l'archive à l'aide d'un script python par exemple.

La valeur du useragent à trouver est : « **Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:35.0) Gecko/20100101 Firefox/35.0** »

En changeant le useragent du navigateur, le résultat de la page `stage4.html` afiche :

Download manager

Download stage5

Le fichier stage5.zip est ainsi obtenu.

5 Etape 5 : Transputers

5.1 Découverte du contenu de l'archive

Le fichier stage5.zip est décompressé afin de découvrir son contenu.

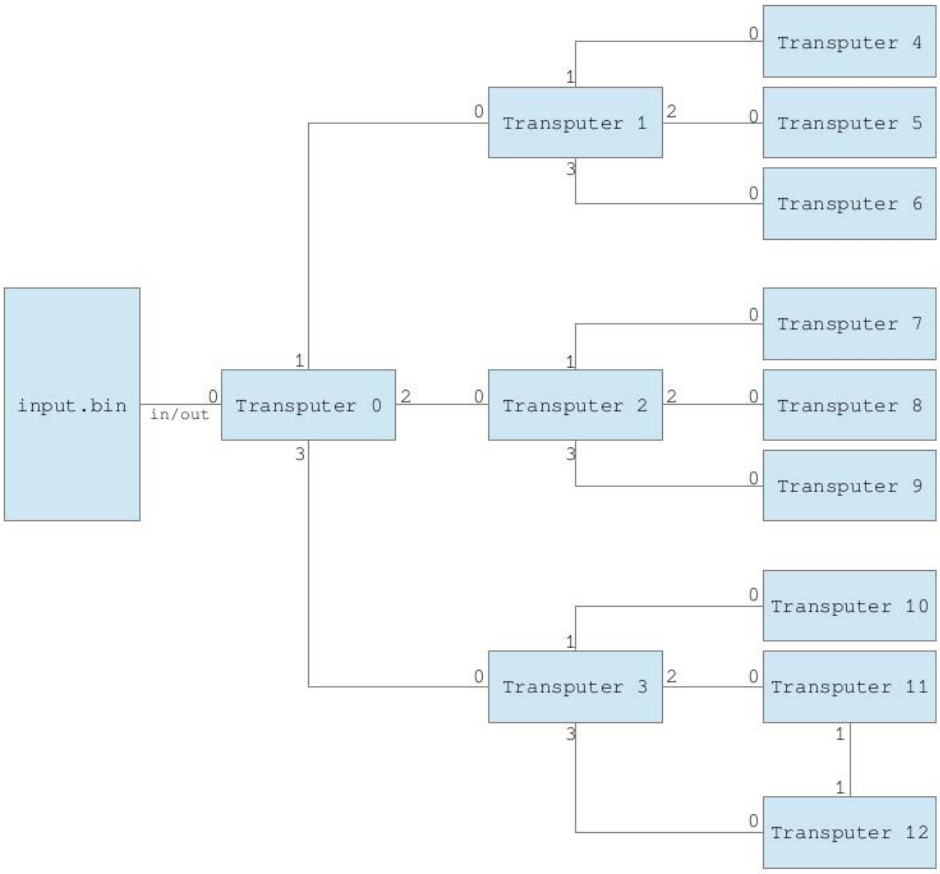
	**Sous Linux** $ unzip stage5.zip

	**Sous Windows** Décompression de l'archive avec 7zip

L'archive était composé des fichiers :

- input.bin
- schematic.pdf

Le fichier schematic.pdf présente l'architecture du ST20, un transputer. Le transputer est une architecture des années 1980 permettant de réaliser des machines en parallèles.



5.2 Documents et outils utiles

Pour réaliser cette épreuve divers documents sont utiles.
Tous sont disponibles sur le site de Michael Brüstle, http://www.transputer.net

- SGS Thomson Datasheets, 42-1440-06 IMS T805 (32-bit floating-point transputer) - February 1996 (http://www.transputer.net/ibooks/dsheets/t805.pdf)
  Le T805 est similaire au ST20. Ce document est utile pour les explications du chapitre “Bootstrap, reset and Analyze”.

- ISBN-10 0-13-929001-X Transputer Reference Manual (72-TRN-006-04) – 1988 (http://www.transputer.net/ibooks/72-trn-006-04/trefm04.pdf)
  Les chapitres 1 et 2 de ce livre expliquent comment les transputers s’échangent les données aux travers des liens, ainsi que la procédure finale de bootstrap.

- Inside the Transputer (D.A.P. Mitchell, J.A. Thompson, G.A. Manson, G.R. Brookes) - 1990 (ISBN 0-632-01689-2) (http://www.transputer.net/iset/isbn-063201689-2/inside.pdf)

- ST20C2/C4 Core Instruction Set Reference Manual - January 1996 (72-TRN-273-01) (http://www.transputer.net/iset/pdf/st20core.pdf)

Les logiciels suivant peuvent-être utiles pour l’analyse :

- st20dis (http://digifusion.jeamland.org/st20dis/)
- IDA Pro (https://www.hex-rays.com/products/ida/) – logiciel comercial
- Hopper (http://www.hopperapp.com/) en version Mac et son plugin ST20 (https://github.com/bSr43/HopperST20C2C4-plugin)

5.3 Analyse du fichier

Suite aux informations des documents ci-dessus, nous savons que le 1^{e} octet indique la longueur du bloc de données de boot du Transputer0. Ce bloc fait 248 (F8h) octets de long.

Le code du Transputer0 a pour rôle d’initialiser ses 3 fils (T1, T2, T3), qui eux même démarreront leur propre fils. Chaque fils reçoit un code de démarrage (bootcode).

La structure bootCode envoyé à chaque Transputer est représentée ainsi

```
Struct bootCode {
        uint32_t length;
        uint32_t linkAddr;
        uint32_t startCode;
}
```

Avec ces informations, on peut découper le fichier ainsi :

Début	Fin	
0	0	Longueur du Transputer0
0001	00F8	Transputer0
00F9	0104	Structure bootCode du Transputer1
0105	0175	Transputer1
0176	0181	Structure bootCode du Transputer2

0182	01F2	Transputer2
01F3	01FE	Structure bootCode du Transputer3
01FF	026F	Transputer3
0270	027B	Structure bootCode du Transputer4
027C	0287	
0288	02AA	Transputer4
02AD	02B8	Structure bootCode du Transputer5
02B9	02C4	
02C5	02E7	Transputer5
02EA	02F5	Structure bootCode du Transputer6
02F6	0301	
0302	0324	Transputer6
0327	0332	Structure bootCode du Transputer7
0333	033E	
033F	0361	Transputer7
0364	036F	Structure bootCode du Transputer8
0370	037B	
037C	039E	Transputer8
03A1	03AC	Structure bootCode du Transputer9
03AD	03B8	
03B9	03DB	Transputer9
03DE	03E9	Structure bootCode du Transputer10
03EA	03F5	
03F6	0418	Transputer10
sss041B	0426	Structure bootCode du Transputer11
0427	0432	
0433	0455	Transputer11
0458	0463	Structure bootCode du Transputer12
0464	046F	
0470	0492	Transputer12
0495	04A0	Structure bootCode destinée à l'exécution du transputer4
04A1	04AC	(code exécuté par le transputer une fois booté)
04AD	04B8	
04B9	04BD	Wrapper fonction in
04BF	04C3	Wrapper fonction out
04C5	04F7	**Transputer4 : code exécuté**
04FD		Structure bootCode destinée à l'exécution du transputer5
0509		(code exécuté par le transputer une fois booté)
0515		
0521	0525	Wrapper fonction in
0527	052B	Wrapper fonction out
052D	055F	**Transputer5 : code exécuté**
0565		Structure bootCode destinée à l'exécution du transputer6
0571		(code exécuté par le transputer une fois booté)
057D		
0589	058D	Wrapper fonction in
058F	0593	Wrapper fonction out
0595	0605	**Transputer6 : code exécuté**
0609		Structure bootCode destinée à l'exécution du transputer7
0615		(code exécuté par le transputer une fois booté)
0621		

062D	0631	Wrapper fonction in
0633	0637	Wrapper fonction out
0639	067F	**Transputer7 : code exécuté**
0685		Structure bootCode destinée à l'exécution du transputer8 (code exécuté par le transputer une fois booté)
0691		
069D		
06A9	06AD	Wrapper fonction in
06AF	06B3	Wrapper fonction out
06B5	0734	**Transputer8 : code exécuté**
0739		Structure bootCode destinée à l'exécution du transputer9 (code exécuté par le transputer une fois booté)
0745		
0751		
075D	0761	Wrapper fonction in
0763	0767	Wrapper fonction out
0769	07A1	**Transputer9 : code exécuté**
07A5		Structure bootCode destinée à l'exécution du transputer10 (code exécuté par le transputer une fois booté)
07B1		
07BD		
07C9	07CD	Wrapper fonction in
07CF	07D3	Wrapper fonction out
07D5	084F	**Transputer10 : code exécuté**
0855		Structure bootCode destinée à l'exécution du transputer11 (code exécuté par le transputer une fois booté)
0861		
086D		
0879	087D	Wrapper fonction in
087F	0883	Wrapper fonction out
0885	08D9	**Transputer11 : code exécuté**
08DD		Structure bootCode destinée à l'exécution du transputer12 (code exécuté par le transputer une fois booté)
08E9		
08F5		
0901	0905	Wrapper fonction in
0907	090B	Wrapper fonction out
090D	0973	**Transputer12 : code exécuté**
0985	0988	Chaîne : "KEY :"
0989	0994	
0995	0995	=23. Longueur du nom de fichier qui va suivre
0996	09AC	Chaîne : " congratulations.tar.bz2"
09AD	3DC9A	Données

5.4 Extraction de l'archive

Suite à l'analyse précédente, le fichier est situé à partir de l'adresse 09AD jusqu'à la fin du fichier. Ce fichier est nommé congratulations.tar.bz2

Vérifions dans un premier temps que l'archive chiffrée est correcte :

Sous Linux

```
$ sha256sum congratulations.tar.bz2
a5790b4427bc13e4f4e9f524c684809ce96cd2f724e29d94dc999ec25e166a81
congratulations.tar.bz2
```

Sous Windows

`sha256sum.exe` congratulations.tar.bz2

```
a5790b4427bc13e4f4e9f524c684809ce96cd2f724e29d94dc999ec25e166a81
congratulations.tar.bz2
```

Le hash retourné correspond bien à celui indiqué dans le pdf. L'archive est donc extraite correctement.

5.5 Traduction du code assembleur

- Algorithme du Transputeur4

```
for (i = 0; i < 12 ; i++)
{
        value = (value + key[i]) & 0xff;
}
return value ;
```

- Algorithme du Transputer5:

```
for (i = 0; i < 12 ; i++)
{
        value = (value ^ key[i]) & 0xff;
}
return value ;
```

- Algorithme du Transputer6:

```
return ((( value & 0x8000 ) >> 15 ) ^ (( value & 0x4000 ) >> 14 ) ^ ( value << 1 )) & 0xFFFF
```

- Algorithme du Transputer7:

```
return (( key[ 0 ] + key[ 1 ] + key[ 2 ] + key[ 3 ] + key[ 4 ] + key[ 5 ] ) ^ ( key[ 6 ] + key[ 7 ] + key[ 8 ] + key[ 9 ] + key[ 10 ] + key[ 11 ] )) & 0xFF
```

- Algorithme du Transputer8:

```
memcpy(var5 + (var4 * 12), key, 12);
var4++;
if (var4 == 4) {
        var4 = 0;
}
for (var2 = 0, var3 = 0; var2 < 4; var2++)
{
        for (var0 = 0, var1 0; var0 < 0x12 ; var0++)
        {
                var1 = (var1 + var5[(var2 * 12) + var0]) & 0xFF;
        }
        var3 = (var3 ^ var1) & 0xFF;
}
return var3;
```

- Algorithme du Transputer9:

```
for (i = 0; i < 12 ; i++)
{
        value = (value ^ (key[i] << (i & 7))) & 0xFF;
}
return value;
```

- Algorithme Transputer10:

```
memcpy(var4 + (var2 * 12), key, 12);
if (++var2 == 4){
        var2 = 0;
}
for (var0 = 0x00, var1 = 0x00; var0 < 0x04; var0++)
{
        var1 = (var1 + var4[var0 * 12]) & 0xFF;
}
var3 = var4[(var1 & 3) * 12 + (((var1 >> 4) % 0xC) & 0xff)];
```

- Algorithme Transputer11:

```
unsigned int x
x = key[ ( pre[ 1 ] ^ pre[ 5 ] ^ pre[ 9 ] ) % 12 ]
memcpy( pre, key, 12 )
return x
```

- Algorithme Transputer12:

```
return key[ ( key[ 0 ] ^ key[ 3 ] ^ key[ 7 ] ) % 12 ]
```

Les transputers 1 à 3, sont ici inutiles.
Le déchiffrement se fait dans le Transputer0 :

```
while (( c = fgetc( fIn )) != EOF ) {
      ChanOut( Out[ 0 ], key, 12 );
      ChanOut( Out[ 1 ], key, 12 );
      ChanOut( Out[ 2 ], key, 12 );
      ChanOut( Out[ 3 ], key, 12 );
      ChanOut( Out[ 4 ], key, 12 );
      ChanOut( Out[ 5 ], key, 12 );
      ChanOut( Out[ 6 ], key, 12 );
      ChanOut( Out[ 7 ], key, 12 );
      ChanOut( Out[ 8 ], key, 12 );

      ChanIn( In[ 0 ], b + 1, 1 );
      ChanIn( In[ 1 ], b + 2, 1 );
      ChanIn( In[ 2 ], b + 3, 1 );
      ChanIn( In[ 3 ], b + 4, 1 );
      ChanIn( In[ 4 ], b + 5, 1 );
      ChanIn( In[ 5 ], b + 6, 1 );
      ChanIn( In[ 6 ], b + 7, 1 );
      ChanIn( In[ 7 ], b + 8, 1 );
      ChanIn( In[ 8 ], b + 9, 1 );

      b[ 1 ] = b[ 1 ] ^ b[ 2 ] ^ b[ 3 ] ^ b[ 4 ] ^ b[ 5 ] ^ b[ 6 ] ^ b[ 7 ] ^ b[ 8 ] ^ b[ 9 ];
      b[ 0 ] = (( key[ i ] << 1 ) + i ) ^ c;
      key[ i ] = b[ 1 ];

      if ( ++i == 12 )  {
            i = 0;
      }
      fputc( b[ 0 ], fOut );
      n++;
   }
```

5.6 Principe du chiffrage

L'algorithme de déchiffrement des données revient à faire une permutation d'octets.
La clé est inconnue.
Cependant l'entête d'un fichier bzip2 est connu :

`.magic:16`	`= 'BZ' signature/magic number`
`.version:8`	`= 'h' for Bzip2 ('H'uffman coding), '0' for Bzip1 (deprecated)`
`.hundred_k_blocksize:8`	`= '1'..'9' block-size 100 kB-900 kB (uncompressed); La valeur standard est '9'`
`.compressed_magic:48`	`= 0x314159265359 (BCD (pi))`
crc:32	

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

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


uint8_t crypted_char[10] = {0xFE, 0xF3, 0x50, 0xDC, 0x81, 0xBC, 0x97, 0x27, 0x89, 0xAC};
uint8_t decrypted_char[10] = {0x42, 0x5A, 0x68, 0x39, 0x31, 0x41, 0x59, 0x26, 0x53, 0x59};


int main(void)
{
	uint16_t c =0;
	uint8_t tmp = 0;
	int i = 0;

	// browse each entry of the arrays
	for(i=0; i<10; i++)
	{
		printf("Resolving '0x%x' char: ", decrypted_char[i]);

		// tests each characters
		for(c=0; c<=0xFF; c++)
		{
			tmp = ((c<<1)+i)^crypted_char[i];
			if (tmp == decrypted_char[i])
			{
				printf("0x%x  ", c);
			}
		}

		printf("\n");
	}
	return 0;
}
```

Le programme nous indique les valeurs suivantes possibles :

```
Resolving '0x42' char: 0x5e  0xde
Resolving '0x5a' char: 0x54  0xd4
Resolving '0x68' char: 0x1b  0x9b
Resolving '0x39' char: 0x71  0xf1
Resolving '0x31' char: 0x56  0xd6
Resolving '0x41' char: 0x7c  0xfc
Resolving '0x59' char: 0x64  0xe4
Resolving '0x26' char: 0x7d  0xfd
Resolving '0x53' char: 0x69  0xe9
Resolving '0x59' char: 0x76  0xf6
```

5.7 Brute-Force

A partir de ces valeurs, nous constatons que pour trouver la clé, nous avons 2^10 possibilités.
Un fichier contenant toutes les valeurs possibles des clés est créé.

La valeur de la clé change tous les 12 octets.
Déchiffrer l'intégralité de l'archive pour savoir si elle est correcte est un travail très long.
Plusieurs solutions étaient possibles :

- Faire du brute force sur une machine, avec l'intégralité de la base
- Faire du brute force sur plusieurs machine, avec chacune une partie de la base. Cette méthode pouvait être réalisée en local, ou à plus grande échelle en louant les service du cloud d'Amazon. Le coût aurait été d'une trentaine d'euros.

Une autre solution existe : l'archive peut-être déchiffrée partiellement.
En effet sur une archive bzip2, après plusieurs tests empiriques, on contente que dans les 64 premiers octets, il y a une grande plage d'octets ayant pour valeurs 'FFh'.

Pour être certain d'avoir bien décodé les données, il faut au moins 2 permutations complètes de clés, donc nous pouvons analyser les données après le 23^{e} octets. Le déchiffrage des données se fera sur les 32 premiers octets.
Le test consistera à vérifier pour chaque clé, si les octets 24 à 31 sont égaux à 'FFh'.

La clé permettant de déchiffrer l'archive est la suivante : "**5ed49b7156fce47de976dac5**"

6 Étape 6 : Tout est question d'image

6.1 Découverte du contenu de l’archive

Le fichier obtenu à l'étape précédente est nommé congratulations.tar.bz2.

Une analyse du fichier confirme que le fichier est bien une archive.

	**Sous Linux** $ file congratulations.tar.bz2 congratulations.tar.bz2: bzip2 compressed data, block size = 900k

Décompressons l'archive :

	**Sous Linux**
	$ bzip2 -d congratulations.tar.bz2
	$ tar -xvf congratulations.tar
	
	congratulations.jpg

	Sous Windows Décompression de l’archive avec 7zip

L'archive contient un seul fichier : congratulations.jpg

6.2 Analyse de l'image JPEG

Après vérification, le fichier congratulations.jpg est bien une image

	**Sous Linux** $ file congratulations.jpg congratulations.jpg: JPEG image data, JFIF standard 1.01

	Sous Windows Le gestionnaire de fichier Total Commander, permet de visualiser le contenu d’un fichier en mode hexadécimal. ffd8 ffe0 0010 **4a46 4946** 0001 0101 0059 ......**JFIF**.....Y L'entête correspond bien au format JPEG

L'image JPEG une fois affichée est bien ce qu'elle prétend être : une image.

Une analyse sous GIMP ne révélant rien d'exploitable, il faut analyser le format JPEG.

6.2.1 Étude du format JPEG

En recherchant sur Google la structure du format JPEG, un lien sur wikipedia est proposé :
http://en.wikipedia.org/wiki/JPEG
Ce lien comporte une information très utile : La fin d'un fichier JPEG est marquée par les octets FF D9.

Avec un éditeur hexadécimal, il est facile de constater que d'autres données existent après les octets FFD9.
Cette hypothèse est vérifiée avec l'utilitaire python hachoir-subfile.

	**Sous Linux** 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)

L'image JPEG est en fait constituée de 2 fichiers :

- une image JPEG, de l'octet 0 à 55247.
- une archive bizp2, de l'octet 55248 à 252569

6.2.2 Extraction du fichier caché

Hachoir-subfile n'a pas pu extraire l'archive bzip2 automatiquement.
L'extraction est donc manuelle avec l'aide de l'utilitaire dd

L'extraction de l'archive bzip2 est réalisé avec l'aide de la commande dd.

	**Sous Linux** dd bs=55248 skip=1 if=congratulations .jpg of=file-0002.bz2

6.3 Analyse de l'archive bzip2

Après vérification le fichier file-002.bz2 est bien une archive bzip2.

	**Sous Linux** $ file file-002.bz2 file-002.bz2: bzip2 compressed data, block size = 900k

	Sous Windows Le gestionnaire de fichier Total Commander, permet de visualiser le contenu d'un fichier en mode hexadécimal. **425a 6839** 3141 5926 5359 beec b4d2 0092 BZh91AY&SY..... L'entête correspond bien au format bzip2

L'archive est ensuite décompressée.

	**Sous Linux** $ bzip2 -d file-002.bz2

	Sous Windows Décompression de l'archive avec 7zip

Un fichier est obtenu : file-002. Ce fichier ne possédant pas d'extensions, il doit être analysé pour en déterminer le type.

	**Sous Linux** $ file file-0002 file-0002: POSIX tar archive (GNU)

	Sous Windows Le gestionnaire de fichier Total Commander, permet de visualiser le contenu d'un fichier en mode hexadécimal. Sur les premiers octets est indiqué le nom du fichier à obtenir après décompression : **congratulations.png**. La structure de cette archive fait penser à une archive de format tar

Le fichier file-002 est renommé en file-002.tar.
Son contenu peut être désormais extrait.

	**Sous Linux** $ tar -xvf file-0002.tar congratulations.png

	Sous Windows

	Décompression de l’archive avec 7zip

Un fichier congratulations.png est obtenu.

6.4 Analyse de l'image PNG

Après vérification, le fichier congratulations.png est bien une image

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

	Sous Windows Le gestionnaire de fichier Total Commander, permet de visualiser le contenu d’un fichier en mode hexadécimal. 8950 4e47 0d0a 1a0a 0000 000d 4948 4452 .PNG.....IHDR L'entête correspond bien au format PNG

L'image PNG une fois affichée est bien ce qu'elle prétend être : une image.

Dans cette nouvelle image, on constate que le texte en bas de l'image a changé « … deux derniers petits efforts ? »

Une analyse sous GIMP ne révélant rien d'exploitable, il faut analyser le format PNG.

6.4.1 Étude du format PNG

L'utilitaire pngcheck est utilisé pour analyser le fichier PNG.

	Sous Linux & Windows

```
$ pngcheck -vtf congratulations.png

File: congratulations.png (197557 bytes)
  chunk IHDR at offset 0x0000c, length 13
    636 x 474 image, 32-bit RGB+alpha, non-interlaced
  chunk bKGD at offset 0x00025, length 6
    red = 0x00ff, green = 0x00ff, blue = 0x00ff
  chunk pHYs at offset 0x00037, length 9: 3543x3543 pixels/meter (90 dpi)
  chunk tIME at offset 0x0004c, length 7: 27 Feb 2015 13:40:19 UTC
  chunk sTic at offset 0x0005f, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x013a2, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x026e5, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x03a28, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x04d6b, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x060ae, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x073f1, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x08734, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x09a77, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x0adba, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x0c0fd, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x0d440, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x0e783, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x0fac6, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x10e09, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x1214c, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x1348f, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x147d2, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x15b15, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x16e58, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x1819b, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x194de, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x1a821, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x1bb64, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x1cea7, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x1e1ea, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x1f52d, length 4919:  illegal reserved-bit-set chunk
  chunk sTic at offset 0x20870, length 38:  illegal reserved-bit-set chunk
  chunk IDAT at offset 0x208a2, length 8192
    zlib: deflated, 32K window, maximum compression
  chunk IDAT at offset 0x228ae, length 8192
  chunk IDAT at offset 0x248ba, length 8192
  chunk IDAT at offset 0x268c6, length 8192
  chunk IDAT at offset 0x288d2, length 8192
  chunk IDAT at offset 0x2a8de, length 8192
  chunk IDAT at offset 0x2c8ea, length 8192
```

```
  chunk IDAT at offset 0x2e8f6, length 6827
  chunk IEND at offset 0x303ad, length 0
ERRORS DETECTED in congratulations.png
```

Cet analyse révèle que l'image PNG est constituée de plusieurs « morceaux » ou « chunks ». Certains morceaux renvoient des erreurs. Ces morceaux possèdent tous le nom **sTic.** Ce nom est trop proche de celui du challenge, pour que cela soit un hasard..
Par acquis de conscience, quelques lectures seront indispensables afin de connaître les morceaux obligatoires :

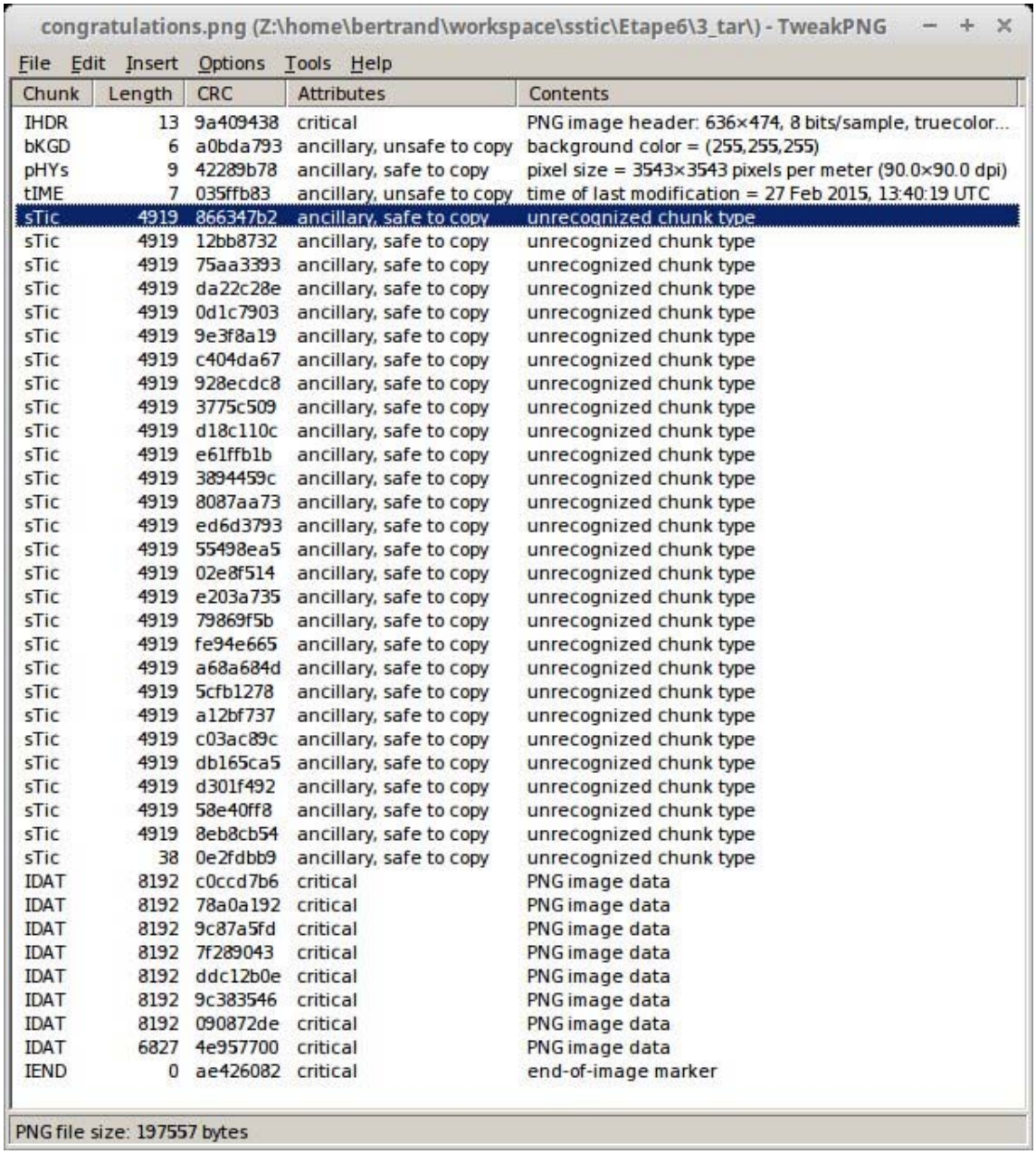
- http://www.w3.org/TR/PNG-Chunks.html
- http://www.libpng.org/pub/png/spec/register/pngreg-1.4.6-pdg.html
- http://www.libpng.org/pub/png/spec/register/pngext-1.4.0-pdg.html

Les morceaux **sTic** ont été rajouté pour les besoins du challenge.

6.4.2 Extraction des morceaux sTic

Le freeware TweakPNG est utilisé pour analyser et extraire le contenu du fichier PNG.

Bien que cet outil soit écrit pour windows, il est possible de l'utiliser sous Linux avec l'aide de Wine.



L'extraction des morceaux est réalisée un à un. Chaque morceaux est enregistré dans un fichier cxx ou xx sert de compteur.

6.4.3 Reconstitution du fichier

Chaque fichier fait 4923 octets à l'exception du 28ième qui fait 42 octets.

Chaque fichier commence par le nom de son morceau (chunk). Partant de l'hypothèse qu'une archive (un fichier) doit être reconstituée par concaténation de chacun des morceaux extraits, les 4 octets "sTiC" sont inutiles.

0000000: **7354 6963** 789c 84b6 7b38 13ee fb38 3ecc **sTic**x...{8...8>.

Pour enlever les 4 premiers octets, dd sera utilisé :

Sous Linux

```
dd bs=4 skip=1  if=c01.chunk of=c01.chunk_fixed
```

L'archive est recrée en concaténant tous les fichiers dont le 4 premiers octets ont été retirés (les fichiers *.chunk_fixed)

Sous Linux

```
cat c01.chunk_fixed c02.chunk_fixed c03.chunk_fixed c04.chunk_fixed c05.chunk_fixed
c06.chunk_fixed c07.chunk_fixed c08.chunk_fixed c09.chunk_fixed c10.chunk_fixed
c11.chunk_fixed c12.chunk_fixed c13.chunk_fixed c14.chunk_fixed c15.chunk_fixed
c16.chunk_fixed c17.chunk_fixed c18.chunk_fixed c19.chunk_fixed c20.chunk_fixed
c21.chunk_fixed c22.chunk_fixed c23.chunk_fixed c24.chunk_fixed c25.chunk_fixed
c26.chunk_fixed c27.chunk_fixed c28.chunk_fixed > file_created
```

6.4.4 Analyse du fichier reconstitué

Ne sachant pas le type de fichier obtenu, on tente une détection du type :

Sous Linux

```
$ file file_created

file_created: data
```

Avec un éditeur hexadécimal, on peut voir que l'entête du fichier commence par 78 9c.
En faisant une recherche sur Google, on peut facilement découvrir que cet entête correspond au format zlib.

Le fichier file_created est renommé en file_created.zlib

Le script python ci-dessous est utilisé pour décompresser l'archive :

```
import zlib
import sys

filename = "file_created.zlib"
with open(filename, 'r') as compressed:
        with open(filename + '-decompressed', 'w') as expanded:
                data = zlib.decompress(compressed.read())
                expanded.write(data)
```

6.4.5 Recherche du type du fichier

Le fichier décompressé ne portant pas d'extension, il faut analyser celui-ci.

	**Sous Linux** $ file file_created-decompressed file_created-decompressed: bzip2 compressed data, block size = 900k

Le fichier obtenu est renommé en file.bzip2. Il ne reste plus qu'à la décompresser.

	**Sous Linux** $ bzip2 -d file.bzip2 bzip2: Can't guess original name for file.bzip2 -- using file.bzip2.out

Le fichier extrait est renommé en file.bzip2.out
Il faut là encore, analyser le type du fichier

	**Sous Linux** $ file file.bzip2.out file.bzip2.out: POSIX tar archive (GNU)

Le fichier est un tar. file.bzip2.out est renommé en file.tar
On le décompresse

	**Sous Linux** $ tar xvf file.tar congratulations.tiff

Un fichier est obtenu : congratulations.tiff

6.5 Analyse de l'image TIFF

Après vérification, le fichier congratulations.tiff est bien une image

	**Sous Linux** $ file congratulations.tiff congratulations.tiff: TIFF image data, little-endian

	Sous Windows Le gestionnaire de fichier Total Commander, permet de visualiser le contenu d'un fichier en mode hexadécimal. 0000000: **4949** 2a00 0800 0000 0900 0001 0300 0100 **II***..... L'entête correspond bien au format TIFF

L'image TIFF une fois affichée est bien ce qu'elle prétend être : une image.

Dans cette nouvelle image, on constate que le texte en bas de l'image a changé « … trois derniers petits efforts ? »

Une analyse sous GIMP ne révélant rien d'exploitable, il faut analyser le format TIFF.

6.5.1 Etude du format TIFF

Une analyse avec un éditeur hexadécimal révèle certaines informations :

```
0000140: 0100 0000 0000 0000 0000 0100 0101 0001  ................
0000150: 0100 0101 0001 0100 0101 0001 0100 0101  ................
0000160: 0001 0100 0101 0001 0100 0101 0001 0100  ................
```

Dans le format TIFF, chaque pixel est encodé sur 3 octets : Rouge, Vert et Bleu.
Le début du fichier devant correspondre à la bordure noire, on pourrait s'attendre à avoir un noir dont chaque pixel aurait pour valeur 0 {0,0,0}.
Hors dans notre cas, certains octets sont à 1.
Ceci fait penser à une technique de stéganographie : l'utilisation des bits de poids faibles d'une image (ou encodage LSB)

6.5.2 Stéganographie : Bits de poids faibles

Après une recherche sur google avec les mots clés « python lsb PILgetdata » les liens suivants sont retournés:
https://ctfwriteups.wordpress.com/tag/steganography/
http://delograND.blogspot.fr/2012/10/hackyou-ctf-steg-300.html
https://stevendkay.wordpress.com/2009/10/07/image-steganography-with-pil/

le code python ci-dessous est à utiliser avec python3.

```
from PIL import Image

im = Image.open("congratulations.tiff")
bins = ""
```

```
for pix in im.getdata():
   bins += (str(pix[0] & 1) + str(pix[1] & 1))

with open("data_extracted", 'wb') as img:
   for i in range(0, len(bins), 8):
      try:
         img.write(bytes([int(bins[i:i+8], 2)]))
      except:
         continue
```

Ce script permet d'obtenir un fichier data_extracted
Le fichier étant de type inconnu, il faut l'analyser.

Sous Linux
$ file data_extracted

data_extracted: bzip2 compressed data, block size = 900k

Le fichier obtenu est renommé en data.b2. Il ne reste plus qu'à la décompresser.

Sous Linux
$ bzip2 -d data.bz2

bzip2: data.bz2: trailing garbage after EOF ignored

Le fichier data est extrait. Là encore, il faut l'analyser.

Sous Linux
$ file data

data: POSIX tar archive (GNU)

Le fichier data est renommé en data.tar. On le décompresse :

Sous Linux
$ tar xvf data.tar

congratulations.gif

Un fichier est obtenu : congratulations.gif

6.6 Analyse de l'image GIF

Après vérification, le fichier congratulations.gif est bien une image

Sous Linux
$ file congratulations.gif

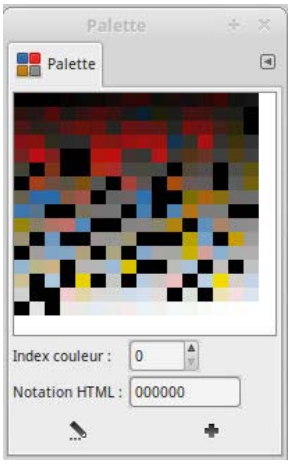
congratulations.gif: GIF image data, version 89a, 636 x 474

	Sous Windows Le gestionnaire de fichier Total Commander, permet de visualiser le contenu d'un fichier en mode hexadécimal. **4749 4638 3961** 7c02 da01 e7ff 0000 0000 **GIF89a**|..... L'entête correspond bien au format GIF

L'image GIF une fois affichée est bien ce qu'elle prétend être : une image.

Dans cette nouvelle image, on constate que le texte en bas de l'image a changé « … quatre derniers petits efforts ? »

6.6.1 Gimp



Les fichiers GIF ont la particularité de disposer d'une palette de couleur intégrée au fichier.
Le fichier peut indexer jusqu'à 256 descriptions de couleurs (R,G,B) stockées dans une table (ou palette) de couleurs appelée look-up table (ou LUT) en anglais.

Ainsi au lieu d'utiliser 3 octets pour décrire chaque pixels, 1 seul est désormais utilisé. Cet octet renvoie à la donnée de la palette correspondant à cette valeur.

Pour afficher la palette des couleurs sous Gimp, il faut cliquer sur la barre de Menu > Fenêtres > Fenêtres ancrables > Palette des couleurs indexées.

La palette des couleurs utilisée sur l'image apparue, il faut sélectionner chaque index de couleur, et changer sa couleur d'origine en une autre (par exemple en rouge)

En choisissant l'index 3, et en changeant sa couleur en rouge par exemple (code FF0000), l'adresse email devient visible.

Cette solution était la plus rapide.

Les personnes les plus motivées ont pu découvrir que la couleur de chaque lettre était codée sur son propre index. Il y a donc autant de couleurs à modifier, que de lettres.
Sur l'image ci-dessous, la couleur verte, non existante sur l'image d'origine a été utilisée afin de montrer les index modifiés.

Les index à modifier sont les numéros : 31, 47, 67, 68, 81, 83, 84, 90, 93, 96, 102, 105, 109, 117, 121, 122, 130, 131, 135, 137, 142, 143, 149, 150, 155, 161, 163, 164, 168, 169, 170, 173, 176, 181,

183, 195, 200, 201, 204, 206, 209, 211, 213, 215, 219, 225, 226, 234, 236, 240, 242.

L'adresse email @challenge.sstic.org que nous devient trouver en début de challenge est trouvée.

6.6.2 Liste des outils utilisés pour résoudre cette épreuve

	**Logiciels spécifiques Linux** Pngcheck - http://www.libpng.org/pub/png/apps/pngcheck.html

	**Logiciels spécifiques Windows** Pngcheck - http://www.libpng.org/pub/png/apps/pngcheck.html Tweakpng - http://entropymine.com/jason/tweakpng/

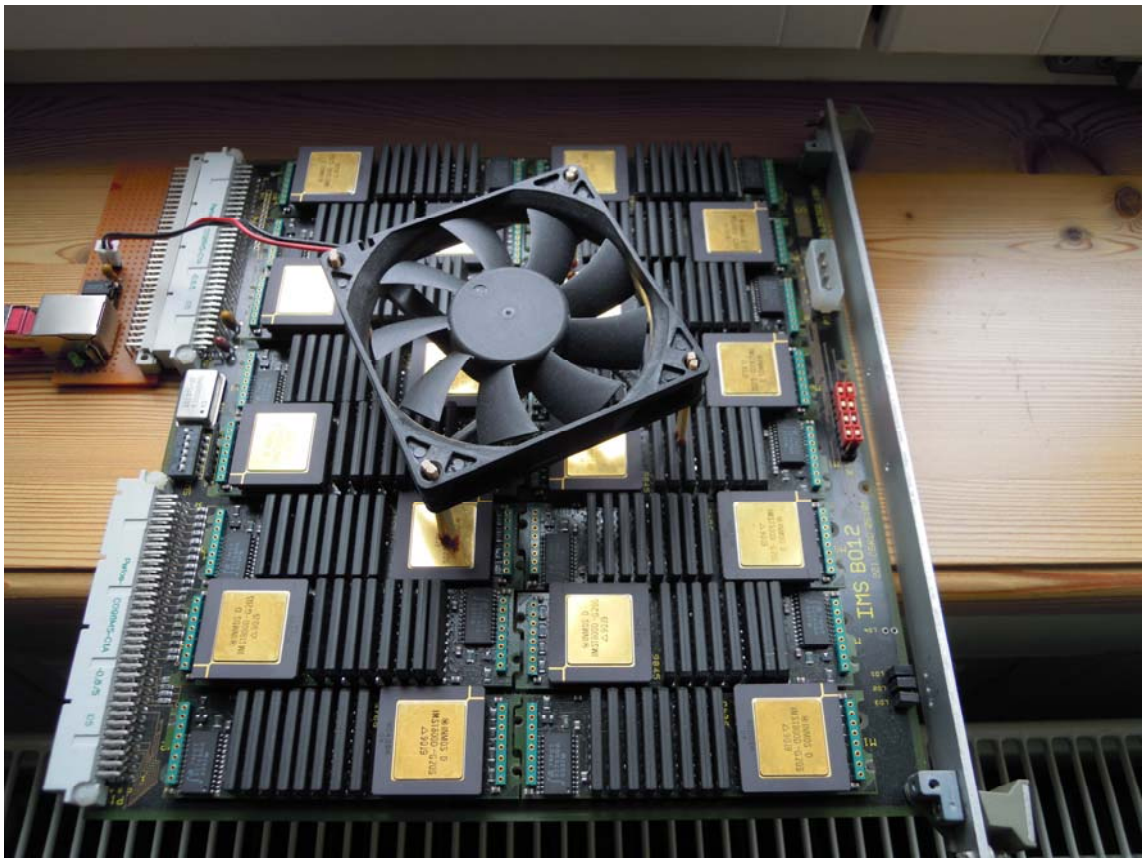
7 Remerciements

Je tiens à remercier particulièrement le concepteur du challenge ainsi que l'ensemble du comité d’organisation du SSTIC.

Un grand merci également à Raphaël Rigo pour ses encouragements ainsi qu’à Vincent Bénony pour son logiciel Hopper ainsi que pour ses explications, obtenues après réussites de l’épreuve.

Je tiens également à remercier chaleureusement Michael Brüstle pour les documents communiqués, ainsi que pour l’émulateur de transputer qu’il m’a fourni. J’ai ainsi pu valider mon portage assembleur vers C en étant sûr de la validité des résultats.

Une fois l’épreuve terminée, Michael fut motivé par mes résultats lors du brute-force et a utilisé un transputer T800, très proche du ST20, pour faire tourner l’image fournie.
La partie brute-force a mis 67 minutes, sur une machine à 20Mhz !
Voici 2 photos d’un transputer T800 que nous souhaitions partager, pour tous ceux qui ont aimé la découverte de cette technologie.




IMS B012
221 CBRD 061-05
INMOS D
IMST800D-G20S
9019
ND439K
UK
NC002K
9045
74FCT
827ASOA
9037M