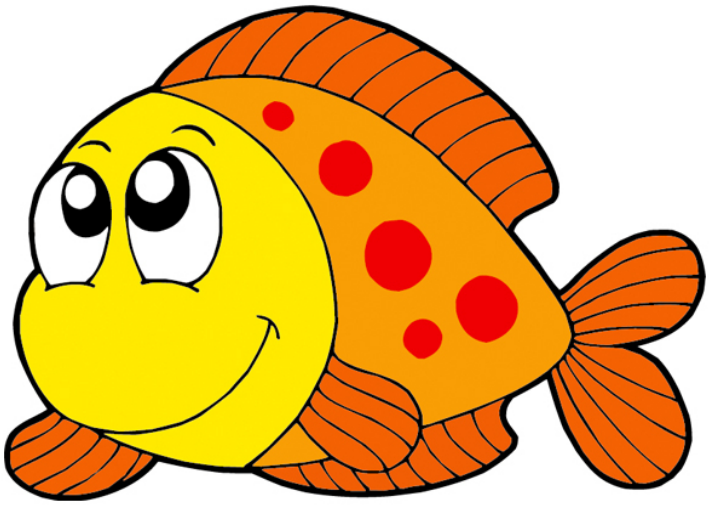
Solution challenge SSTIC 2015

0-Rulez

Challenge plutôt facile cette année :

```
wget http://static.sstic.org/challenge2015/chlg-2015
openssl enc -d -aes-128-ecb -in chlg-2015 -out fini.png -pass pass:sstic
```



Solution du challenge SSTIC 2015. 0-Rulez, ou presque : utiliser autant que possible bash, sed, awk, dd et xxd.

Stage 1

On utilise photorec sur l'image de la sdcard afin d'extraire les fichiers. Le fichier build.sh indique qu'il s'agit d'un dongle USB rubber ducky.

On récupère le script perl de reverse du ducky vers du texte, on l'exécute et on décode tout ça (extraction des commandes batch, traitement des retours à la ligne utf16, décodage base64, passage en utf8, nettoyage, décodage base64) :

```
$ wget "https://ducky-decode.googlecode.com/svn-history/r6/trunk/ducky-decode.pl"

$ perl ducky-decode.pl -f inject.bin > duckyscript.txt

$ cat duckyscript.txt | sed -n -e '/ - e n c/,/ENTER/{/ - e n c/d;/ENTER/d;/SPACE/d;p;}' | sed -e "s,00a0,,g" -e "s, ,,g" | while read l; do echo $l | base64 -d | iconv -f utf16 -t utf-8 ; echo ""; done | sed "s,FromBase64String('\([^']*\)'.*,\n\1,g" | grep -v "^function" | base64 -d > stage2.zip
```

Stage 2

Il s'agit d'une carte openarena. On joue avec openarena en trichant (noclip), dans la console :

```
\devmap sstic

\noclip

\bind g screenshot
```

On retrouve tous les morceaux de clé, on les assemble pour avoir la clé « `9e2f31f78153296b3d9b0ba67695dc7cb0daf152b54cdc34ffe0d35526609fac` »

On déchiffre avec openssl :

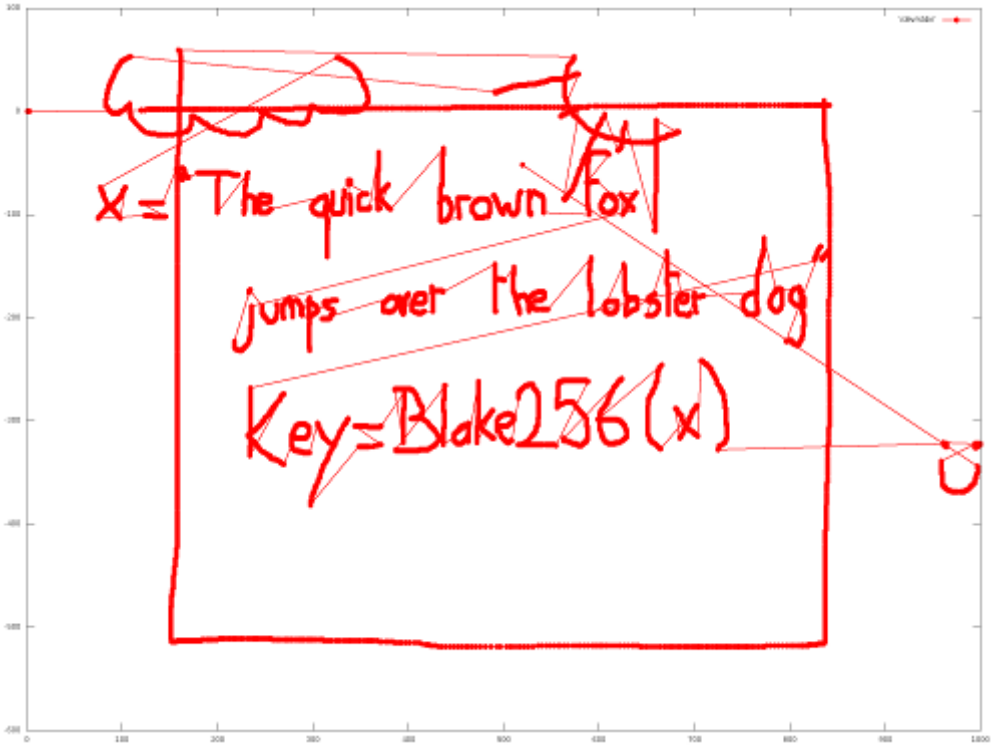
```
$ openssl enc -d -K 9e2f31f78153296b3d9b0ba67695dc7cb0daf152b54cdc34ffe0d35526609fac -iv 535354494332303135 2d537461676532 -aes-256-ofb -in encrypted -out decrypted
```

Pour obtenir le bon sha256, il faut retirer le padding.

Stage 3

Il s'agit d'une capture USB de la souris. On génère une image pdf du tracé de la souris bouton cliqué avec gnuplot (extraction des trames de la souris, nettoyage, décodage hexa vers décimal, passage valeurs relatives en absolues, gnuplot)

```
$ /usr/sbin/tcpdump -r paint.cap -x | grep "0x0000:" | sed -e "s,.*:\s*\
(....\) \(....\).*,\1\2,g" -e "s,\(..\)\(..\)\(..\)..,0x\1 0x\2 0x\3,g" |
awk '{ x=strtonum($2); y=strtonum($3); if(x>128) x=x-256; if(y>128) y=y-256;
xc=xc+x; yc=yc-y; print xc,yc,strtonum($1)}' | grep -v "0$" | gnuplot -e
"set terminal pdf size 60cm, 40cm; set output 'out.pdf'; set style line 1 lt
1 lw 2 pt 7 ps 1.5; plot '/dev/stdin'  with linespoints ls 1"
```



On télécharge, compile, execute Blake256:

```
$ wget https://github.com/veorq/BLAKE/raw/master/blake256.c ; wget
https://github.com/veorq/BLAKE/raw/master/blake.h

$ gcc -o blake256 blake256.c

$ echo -n "The quick brown fox jumps over the lobster dog" | ./blake256
/dev/stdin
```

On trouve que la clé est « `66c1ba5e8ca29a8ab6c105a9be9e75fe0ba07997a839f-feae9700b00b7269c8d` »

On utilise la lib crypto++ pour déchiffrer le Serpent-1-CBC-CTS, source fourni en annexe 1 :

```
$ xxd -i encrypted > encrypted.h

$ g++ -o serpent serpent.cpp -I/usr/include/crypto++ -lcryptopp -lpthread
-DNDEBUG

$ ./serpent | xxd -r -p > stage4.zip
```

Stage 4

Il s'agit d'un fichier HTML avec du javascript obfusqué.

On place un breakpoint dans Firefox sur le return du code, puis après un passage dans jsdetox le code est plus lisible. Un nettoyage manuel, renommage des variables permet une meilleure lecture.

Le code effectue le déchiffrement du contenu de la variable data en utilisant comme clé et iv le contenu du user-agent. Seule la partie entre parenthèses du user-agent est utilisée.

Le HTML indique « `Lucida Grande` » qui est une police MacOS X.

Le code fait référence à « `preferences.xul` » il s'agit vraisemblablement de firefox.

L'API crypto subtle est apparue dans firefox 34.

En prenant les quatre dernières versions de MacOSX, les 12 versions de firefox entre 34 et 37.0.1 seuls 48 valeurs de user-agents sont possible. L'exécution du code javascript (en annexe 2) montre que le user agent attendu est « `a (Macintosh; Intel Mac OS X 10.6; rv:35.0) b` ».

NB : Le hash javascript présent dans le code du challenge est faux, il a été effectué avec le padding, or l'api crypto retire le padding.

Stage 5

Le fichier input.bin est une image de code ST20. Une rétro-analyse avec IDA permet après un travail laborieux d'écrire l'algorithme de chiffrement/déchiffrement en C fourni en annexe 3.

La charge encrypted se trouve à la fin du fichier :

```
$ dd if=input.bin of=encrypted bs=2477 skip=1
```

Lors du déchiffrement des 12 premiers octets, la clé est utilisée sans transformation :

```
out = (2*key[i] + i) ^ in
```

Les 7 bits de poids faible de chaque octet de la clé sont vulnérables à une attaque par clair connu. Le fichier étant un tar.bz2, un certain nombre de bits est connu :

```
BZh.1AY&SY..
```

Le quatrième octet est le taux de compression. Grâce à la parité, on sait que ce taux est 1, 3, 5, 7 ou 9. Les 11ème et 12ème octets doivent être brute-forcés.

Taille de l'espace de brute-force :

```
2e10*5*1e16 soit environ 335 millions de tests
```

L'algorithme étant relativement lent, il est impératif de discriminer rapidement les échecs, sans déchiffrer tout. Un test doit être effectué sur les bits du deuxième tour. Si le fichier est peu compressible la table d'Huffman aura peu de symboles et il y aura donc beaucoup de 0xFF dans le header du tar.bz2. Afin d'éviter d'éviter les faux négatifs, seuls quelques octets choisis ont été testés.

Après 3 min 30 de brute-force, la clé est « `5ed49b7156fce47de976dac5` » et la charge est déchiffrée.

Stage 6

Partie 1

Il s'agit d'une image JPEG. Un fichier bzip2 a été concaténé à la fin. On recherche le header bzip2 et on l'extrait :

```
$ dd if=congratulations.jpg of=stage6.2.tar.bz2 bs=$(grep -Pabo "BZh"
congratulations.jpg | cut -d":" -f1) skip=1
```

Partie 2

Il s'agit d'une image PNG. Des chunks suspects de data de nom sTIc et de taille 0x1377 sont présents. On les extrait, ils paraissent aléatoires, du coup on les décompresse avec zlib :

```
$ for c in $(grep -Pabo "sTic" congratulations.png | cut -d":" -f1); do dd
if=congratulations.png bs=1 skip=$((c+4)) count=$((0x1337)); done | openssl
zlib -d > stage6.3.tar.bz2
```

NB: Sur des mauvaises distributions (!{ge|fu}ntoo) openssl est compilé sans le support du chiffrement par l'algorithme deflate. Il est dans ce cas possible d'utiliser perl en ligne à la place.

```
$ perl -MCompress::Zlib -e 'undef $/; print uncompress(<>)'
```

Partie 3

Il s'agit d'une image TIF. Un passage dans GIMP et une manipulation des niveaux montre qu'il s'agit de données codées sur les bits de poids faible de deux des trois composantes de l'image. On extrait ces bits et on reconstitue le fichier :

```
$ (echo -n "ibase=2; "; xxd  -c 3 -b -s 128 congratulations.tiff | sed
"s,.*: [01]\{7\}\(.\) [01]\{7\}\(.\).*$,\1\2,g" | tr -d "\n"| sed "s,\
(........\),\1;,g"; echo) | bc | awk '{printf "%02x",$1}' | xxd -r -p >
stage6.4.tar.bz2
```

Partie 4

Il s'agit d'une image GIF. La palette GIF contient de nombreuses couleurs mise à `000000` (noir). Si l'on change la couleur noire du fond on voit l'email :

```
$ printf "\xff\x80\x80" | dd of=c.gif conv=notrunc bs=1 count=3 seek=19
```

Une technique plus amusante est d'écraser la palette avec de l'aléa. Il suffit d'écraser un bout du fichier avec de l'aléa pour avoir une réponse psychédélique :

```
$ cp congratulations.gif fini.gif
$ dd if=/dev/urandom of=fini.gif conv=notrunc bs=1 count=768 seek=13
```

Annexe 1 :

```
// serpent.c
// make encrypted.h with: xxd -i encrypted > encrypted.h
// compile with: g++ -o serpent serpent.cpp -I/usr/include/crypto++ -lcryptopp
// -lpthread -DNDEBUG
//
#include <iostream>
using std::cout;
using std::cerr;
using std::endl;

#include <string>
using std::string;

#include <cstdlib>
using std::exit;

#include "cryptlib.h"
using CryptoPP::Exception;

#include "hex.h"
using CryptoPP::HexEncoder;

#include "filters.h"
using CryptoPP::StringSink;
using CryptoPP::StringSource;
using CryptoPP::StreamTransformationFilter;

#include "serpent.h"
using CryptoPP::Serpent;

#include "modes.h"
using CryptoPP::CBC_CTS_Mode;

#include "secblock.h"
using CryptoPP::SecByteBlock;

#include "encrypted.h"

int main(int argc, char* argv[])
{
	byte key[32] = {0x66,0xc1,0xba,0x5e,0x8c,0xa2,0x9a,0x8a,0xb6,0xc1,0x05,
			0xa9,0xbe,0x9e,0x75,0xfe,0x0b,0xa0,0x79,0x97,0xa8,0x39,
			0xff,0xea,0xe9,0x70,0x0b,0x00,0xb7,0x26,0x9c,0x8d};
	byte iv[] = {0x53,0x53,0x54,0x49,0x43,0x32,0x30,0x31,
			0x35,0x2d,0x53,0x74,0x61,0x67,0x65,0x33};

	string encoded, recovered;
	string cipher((char*)encrypted, sizeof(encrypted));

	try
	{
		CBC_CTS_Mode< Serpent >::Decryption d;
		d.SetKeyWithIV(key, sizeof(key), iv);

		StringSource ss1(cipher, true,
			new StreamTransformationFilter(d,
				new StringSink(recovered)
			)
		);
```

```
		encoded.clear();
		StringSource ss2(recovered, true,
			new HexEncoder(
				new StringSink(encoded)
			)
		);
		cout << encoded << endl;
	}
	catch(const CryptoPP::Exception& e)
	{
		cerr << e.what() << endl;
		exit(1);
	}

	return 0;
}
```

Annexe 2

```
<html>
<head>
<style>
	* { font-family: Lucida Grande,Lucida Sans Unicode,Lucida
Sans,Geneva,Verdana,sans-serif; text-align:center; }
	#status { font-size: 16px; margin: 20px; }
	#status a { color: green; }
	#status b { color: red; }
</style>
</head>
<body>
	<script>
		var data = //... trop long pour ce document
	//Hash is bad, remove padding but hash is calculated on padde data...
		//var hash = "08c3be636f7dffd91971f65be4cec3c6d162cb1c";

		ua_list=[
			"a (Macintosh; Intel Mac OS X 10.6; rv:34.0) b",
			"a (Macintosh; Intel Mac OS X 10.6; rv:34.0.1) b",
			"a (Macintosh; Intel Mac OS X 10.6; rv:34.0.5) b",
			"a (Macintosh; Intel Mac OS X 10.6; rv:35.0) b",
			"a (Macintosh; Intel Mac OS X 10.6; rv:35.0.1) b",
			"a (Macintosh; Intel Mac OS X 10.6; rv:36.0) b",
			"a (Macintosh; Intel Mac OS X 10.6; rv:36.0.1) b",
			"a (Macintosh; Intel Mac OS X 10.6; rv:36.0.2) b",
			"a (Macintosh; Intel Mac OS X 10.6; rv:36.0.3) b",
			"a (Macintosh; Intel Mac OS X 10.6; rv:36.0.4) b",
			"a (Macintosh; Intel Mac OS X 10.6; rv:37.0) b",
			"a (Macintosh; Intel Mac OS X 10.6; rv:37.0.1) b",

			"a (Macintosh; Intel Mac OS X 10.7; rv:34.0) b",
			"a (Macintosh; Intel Mac OS X 10.7; rv:34.0.1) b",
			"a (Macintosh; Intel Mac OS X 10.7; rv:34.0.5) b",
			"a (Macintosh; Intel Mac OS X 10.7; rv:35.0) b",
			"a (Macintosh; Intel Mac OS X 10.7; rv:35.0.1) b",
			"a (Macintosh; Intel Mac OS X 10.7; rv:36.0) b",
			"a (Macintosh; Intel Mac OS X 10.7; rv:36.0.1) b",
			"a (Macintosh; Intel Mac OS X 10.7; rv:36.0.2) b",
			"a (Macintosh; Intel Mac OS X 10.7; rv:36.0.3) b",
			"a (Macintosh; Intel Mac OS X 10.7; rv:36.0.4) b",
			"a (Macintosh; Intel Mac OS X 10.7; rv:37.0) b",
			"a (Macintosh; Intel Mac OS X 10.7; rv:37.0.1) b",

			"a (Macintosh; Intel Mac OS X 10.8; rv:34.0) b",
			"a (Macintosh; Intel Mac OS X 10.8; rv:34.0.1) b",
			"a (Macintosh; Intel Mac OS X 10.8; rv:34.0.5) b",
			"a (Macintosh; Intel Mac OS X 10.8; rv:35.0) b",
			"a (Macintosh; Intel Mac OS X 10.8; rv:35.0.1) b",
			"a (Macintosh; Intel Mac OS X 10.8; rv:36.0) b",
			"a (Macintosh; Intel Mac OS X 10.8; rv:36.0.1) b",
			"a (Macintosh; Intel Mac OS X 10.8; rv:36.0.2) b",
			"a (Macintosh; Intel Mac OS X 10.8; rv:36.0.3) b",
			"a (Macintosh; Intel Mac OS X 10.8; rv:36.0.4) b",
			"a (Macintosh; Intel Mac OS X 10.8; rv:37.0) b",
			"a (Macintosh; Intel Mac OS X 10.8; rv:37.0.1) b",

			"a (Macintosh; Intel Mac OS X 10.9; rv:34.0) b",
			"a (Macintosh; Intel Mac OS X 10.9; rv:34.0.1) b",
			"a (Macintosh; Intel Mac OS X 10.9; rv:34.0.5) b",
```

```
                    "a (Macintosh; Intel Mac OS X 10.9; rv:35.0) b",
                    "a (Macintosh; Intel Mac OS X 10.9; rv:35.0.1) b",
                    "a (Macintosh; Intel Mac OS X 10.9; rv:36.0) b",
                    "a (Macintosh; Intel Mac OS X 10.9; rv:36.0.1) b",
                    "a (Macintosh; Intel Mac OS X 10.9; rv:36.0.2) b",
                    "a (Macintosh; Intel Mac OS X 10.9; rv:36.0.3) b",
                    "a (Macintosh; Intel Mac OS X 10.9; rv:36.0.4) b",
                    "a (Macintosh; Intel Mac OS X 10.9; rv:37.0) b",
                    "a (Macintosh; Intel Mac OS X 10.9; rv:37.0.1) b"]

              document['write']('<h' + 1 + '>Down' + 'load' + ' ' + 'manager</h' + 1 +
'>');
              document['write']("<div id=\"status\"><i>loading...</i></div>");
              document['write']("<div stylexxx=\"display:none\"><a target=\"blank\"
href=\"chrome://browser/content/preferences/preferences.xul\">Back to
preferences</a></div>");

              function str2bin(param){
                var ret = [];
                for(var i = 0; i < param['length']; ++i)
                    ret.push(param['charCodeAt'](i));
                return new Uint8Array(ret);
              }

              function hex2bin(param){
                var ret = [];
                for(var i = 0; i < param['length'] / 2; ++i)
                    ret.push(parseInt(param['substr'](i * 2, 2), 16));
                return new Uint8Array(ret);
              }

              function bin2hex(param){
                var ret = '';
                for(var i = 0; i < param.byteLength; ++i) {
                    var ___ = param[i].toString(16);
                    if(___['length'] < 2) ret += 0;
                    ret += ___;
                }
                return ret;
              }

              data_bin=hex2bin(data);

              function fun4(ua){
                    var iv_str = ua.substr(ua.indexOf('(') + 1, 16);
                    var key_str = ua.substr(ua.indexOf(')') - 16, 16);
                    var iv = str2bin(iv_str);
                    var key = str2bin(key_str);
                    var param = { };
                    param['name'] = 'AES-CBC';
                    param['iv'] = iv;
                    param['length'] = key['length'] * 8;
                    window.crypto.subtle.importKey("raw", key, param, false,
['decrypt']).then(function(res1) {
                    window.crypto.subtle.decrypt(param, res1,
data_bin).then(function(res2) {
                          var deciphered = new Uint8Array(res2);
                          //window.crypto.subtle['digest']({name: 'SHA-1'},
deciphered).then(function(res3) {
                          //if(hash == bin2hex(new Uint8Array(res3))) {
                                var prop = { };
                                prop['type'] = 'application/octet-stream';
```

```
                                    var hash = new Blob([deciphered], prop);
                                    var url = URL['createObjectURL'](hash);
                                    document['getElementById']('status')['innerHTML'] =
"<b>UA: " + ua +"</b><br/><a href=\"" + url + "\" download=\"stage5.zip\">download
stage5</a>";
                              //} else {
                              //    document['getElementById']('status')['innerHTML'] =
"<b>Failed to load stage5, bad hash</b>" + bin2hex(deciphered);
                              //}
                              //});
                        }).catch(function(e) {
                              console.log(e);
                        });
                        }).catch(function(e) {
                              console.log(e);
                        });
                  }

                  window['setTimeout'](function fun5() { var my_i;
for(my_i=0;my_i<ua_list.length;my_i++)fun4(ua_list[my_i]); }, 500);
            </script>
</body>
</html>
```

Annexe 3

```c
/* SSTIC challenge stage 5
    indent with : indent -linux -npcs -br -brf -brs -l120 emul.c
    compile with: gcc -O3 -Wall -ggdb3 -o emul emul.c -lssl -fwhole-program
-Wextra -march=native
    note, on some distros use -lcrypto instead of -lssl
*/

#include <string.h>
#include <stdio.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdlib.h>
#include <openssl/sha.h>

typedef unsigned char u8;
typedef unsigned short u16;
typedef unsigned int u32;

static int transp0bzip2(const u8 * key, const u8 * data_in, u8 * data_out, int
size);
static u8 transp1(const u8 * buff);
static u8 transp2(const u8 * buff);
static u8 transp3(const u8 * buff);
static u8 transp4(const u8 * buff);
static u8 transp5(const u8 * buff);
static u8 transp6(const u8 * buff);
static u8 transp7(const u8 * buff);
static u8 transp8(const u8 * buff);
static u8 transp9(const u8 * buff);
static u8 transp10(const u8 * buff);
static u8 transp11(const u8 * buff);
static u8 transp12(const u8 * buff);
static void init(void);
static int bzip2_eof_detect(const u8 * buff, int sz);

static const u8 ref_sha[] =
    { 0x91, 0x28, 0x13, 0x51, 0x29, 0xd2, 0xbe, 0x65, 0x28, 0x09, 0xf5, 0xa1,
0xd3, 0x37, 0x21, 0x1a, 0xff, 0xad, 0x91,
0xed, 0x58, 0x27, 0x47, 0x4b, 0xf9, 0xbd, 0x7e, 0x28, 0x5e, 0xce, 0xf3, 0x21 };

int main(int argc, char *argv[]) {
    if (argc < 2) {
        return -1;
    }
    FILE *f = fopen(argv[1], "r");
    struct stat stat_buf;
    stat(argv[1], &stat_buf);
    int sz = stat_buf.st_size;
    u8 *enc = (u8 *) malloc(sz);
    u8 *dec = (u8 *) malloc(sz);

    fread(enc, sz, 1, f);
    fclose(f);
    /* compression level 9, 7, 5, 3, 1 */
    u8 comp[] = { 0x71, 0x74, 0x73, 0x76, 0x75 };

    u32 j, m, val;
```

```
    for (j = 0; j < sizeof(comp); j++) {
        printf("[-] Brute force compression ratio %i\n", 9 - j * 2);
        for (m = 0; m < 1024; m++) {
            printf("\r[-] Brute force high bits: %i/1024", m);
            fflush(stdout);

            /* default key */
            u8 key[] = { 0x5e, 0x54, 0x1b, 0x71, 0x56, 0x7c, 0x64, 0x7d, 0x69,
0x76, 0x62, 0x31 };

            key[3] = comp[j];

            /* brute force high bits */
            key[0] |= (m & 1) << 7;
            key[1] |= ((m >> 1) & 1) << 7;
            key[2] |= ((m >> 2) & 1) << 7;
            key[3] |= ((m >> 3) & 1) << 7;
            key[4] |= ((m >> 4) & 1) << 7;
            key[5] |= ((m >> 5) & 1) << 7;
            key[6] |= ((m >> 6) & 1) << 7;
            key[7] |= ((m >> 7) & 1) << 7;
            key[8] |= ((m >> 8) & 1) << 7;
            key[9] |= ((m >> 9) & 1) << 7;

            /* brute force high bits */
            for (val = 0; val < 65536; val++) {
                key[10] = val & 0xFF;
                key[11] = (val >> 8) & 0xFF;

                if (transp0bzip2(key, enc, dec, sz) == 0)
                    continue;

                /* printf("\n[+] Valid bzip2 hdr"); */
                if (bzip2_eof_detect(dec, sz)) {
                    printf("\n[+] Valid bzip2 eof\n");
                    SHA256_CTX ctx;
                    u8 digest[32];
                    SHA256_Init(&ctx);
                    SHA256_Update(&ctx, dec, sz);
                    SHA256_Final(digest, &ctx);

                    if (memcmp(digest, ref_sha, 16) == 0) {
                        printf("[+] Found !, write to 'decrypted' with key:");
                        int i;
                        for (i = 0; i < 12; i++) {
                            printf("%02x", key[i]);
                        }
                        printf("\n");
                        FILE *o = fopen("decrypted", "w");
                        fwrite(dec, sz, 1, o);
                        fclose(o);
                    }

                }
            }
        }

    }

    free(enc);
    free(dec);
    return 0;
}
```

```
static int bzip2_hdr_detect(const u8 * buff) {
    /* randomized an upper bits of oriptr are zeros */
    if (buff[14] & 0xF8) {
        return 0;
    }

    /* Only few bits in huffman maps most probably 1 */
    if ((buff[17] & 0x1F) != 0x1F || (buff[18] & 0x80) != 0x80) {
        return 0;
    }
    /* may be a bzip2... */
    return 1;
}

static int bzip2_eof_detect(const u8 * buff, int sz) {
    u8 magics[8][4] = {
        {0x22, 0x9c, 0x28, 0x48},
        {0x24, 0x53, 0x85, 0x09},
        {0x48, 0xa7, 0x0a, 0x12},
        {0x72, 0x45, 0x38, 0x50},
        {0x91, 0x4e, 0x14, 0x24},
        {0x92, 0x29, 0xc2, 0x84},
        {0xc9, 0x14, 0xe1, 0x42},
        {0xe4, 0x8a, 0x70, 0xa1}
    };
    int i;
    for (i = 0; i < 8; i++) {
        if (memcmp(buff + sz - 9, &magics[i][0], 4) == 0) {
            return 1;
        };
    }
    return 0;
}

static int transp0bzip2(const u8 * param_key, const u8 * data_in, u8 *
param_data_out, int size) {
    u8 key[12], *data_out = param_data_out;
    int decoded_sz = 0, i = 0;
    init();
    memcpy(key, param_key, 12);
    while (decoded_sz++ < size) {
        *data_out = (2 * key[i] + i) ^ *data_in;
        if (decoded_sz == 15 && ((*data_out) & 0xF8))
            return 0;
        if (decoded_sz == 18 && ((*data_out) & 0x1F) != 0x1F)
            return 0;
        if (decoded_sz == 19 && ((*data_out) & 0x80) != 0x80)
            return 0;

        data_out++;
        data_in++;

        key[i] = transp1(key) ^ transp2(key) ^ transp3(key);
        i++;
        i %= 12;
        if (decoded_sz == 20 && !bzip2_hdr_detect(param_data_out)) {
            return 0;
        }
    }
    return 1;
}

static u8 transp1(const u8 * buff) {
    return transp4(buff) ^ transp5(buff) ^ transp6(buff);
```

```
}

static u8 transp2(const u8 * buff) {
    return transp7(buff) ^ transp8(buff) ^ transp9(buff);
}

static u8 transp3(const u8 * buff) {
    return transp10(buff) ^ transp11(buff) ^ transp12(buff);
}

static u8 transp4_w1 = 0;
static u8 transp4(const u8 * buff_w2) {
    int i;
    for (i = 0; i < 12; i++) {
        transp4_w1 += buff_w2[i];
    }
    return transp4_w1;
}

static u8 transp5_w1 = 0;
static u8 transp5(const u8 * buff_w2) {
    int i;
    for (i = 0; i < 12; i++) {
        transp5_w1 ^= buff_w2[i];
    }
    return transp5_w1;
}

static u16 transp6_w1 = 0;
static int transp6_init_w3 = 0;    //init
static u8 transp6(const u8 * buff_w4) {
    int i;
    if (transp6_init_w3 == 0) {
        for (i = 0; i < 12; i++) {
            transp6_w1 += buff_w4[i];
        }
        transp6_init_w3 = 1;
    }
    transp6_w1 = ((transp6_w1 & 0x8000) >> 15) ^ ((transp6_w1 & 0x4000) >> 14) ^
(transp6_w1 << 1);
    return (transp6_w1 & 0xFF);
}

static u8 transp7(const u8 * buff_w4) {
    int i;
    u8 w1 = 0, w2 = 0;
    for (i = 0; i < 6; i++) {
        w1 = buff_w4[i] + w1;
        w2 = buff_w4[i + 6] + w2;
    }
    return (w2 ^ w1) & 0xFF;
}

static u8 transp8_buff_w5[48] = { 0 };

static int transp8_w4 = 0;
static u8 transp8(const u8 * buff) {
    int w2;
    memcpy(transp8_buff_w5 + transp8_w4 * 12, buff, 12);
    u8 w3 = 0;

    transp8_w4++;
    transp8_w4 %= 4;
```

```
    for (w2 = 0; w2 < 4; w2++) {
        int w0;
        u8 w1 = 0;
        for (w0 = 0; w0 < 12; w0++) {
            w1 = transp8_buff_w5[w2 * 12 + w0] + w1;
        }
        w3 = (w3 ^ w1);
    }
    return w3;
}

static u8 transp9(const u8 * buff) {
    int i;
    u8 w1 = 0;
    for (i = 0; i < 12; i++) {
        w1 = ((buff[i] << (i & 7)) ^ w1);
    }
    return w1;
}

static u8 transp10_buff_w4[48] = { 0 };

static int transp10_w2 = 0;
static u8 transp10(const u8 * buff) {
    int w0;
    u8 w1 = 0;

    memcpy(transp10_buff_w4 + transp10_w2 * 12, buff, 12);

    transp10_w2++;
    transp10_w2 %= 4;

    for (w0 = 0; w0 < 4; w0++) {
        w1 += transp10_buff_w4[w0 * 12];
    }
    return transp10_buff_w4[((w1 % 4) * 12) + ((w1 >> 4) % 12)];
}

static u8 transp11_buff[12] = { 0 };    //from transp12
static u8 transp11(const u8 * buff) {
    u8 w1 = transp11_buff[1] ^ transp11_buff[5] ^ transp11_buff[9];    //from
transp12
    memcpy(transp11_buff, buff, 12);    //from transp12
    return buff[w1 % 12];
}

static u8 transp12(const u8 * buff) {
    u8 w1 = (buff[0] ^ buff[3] ^ buff[7]) & 0xFF;    //from transp11
    return buff[w1 % 12];
}

static void init(void) {
    transp4_w1 = 0;
    transp5_w1 = 0;
    transp6_w1 = 0;
    transp6_init_w3 = 0;
    memset(transp8_buff_w5, 0, 48);
    transp8_w4 = 0;
    memset(transp10_buff_w4, 0, 48);
    transp10_w2 = 0;
    memset(transp11_buff, 0, 12);
}
```