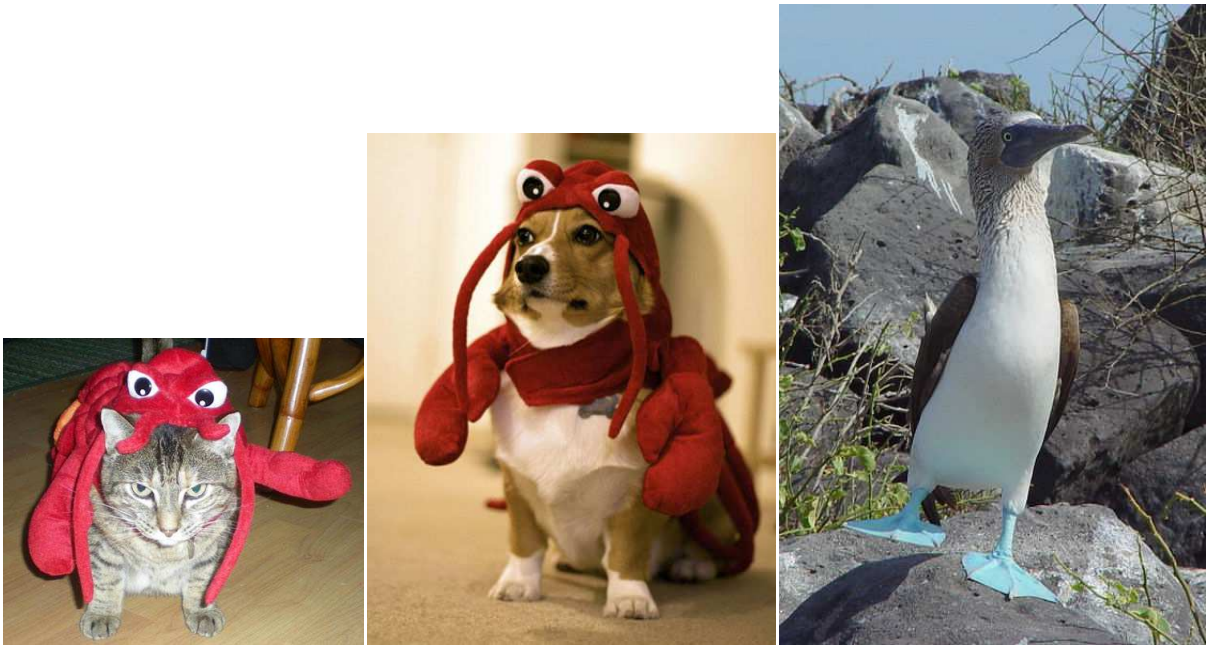
CHALLENGE SSTIC 2012

JEAN SIGWALD



SOMMAIRE

INTRODUCTION

Le challenge SSTIC 2012 consiste à analyser une image disque :

```
$ wget http://static.sstic.org/challenge2012/challenge
$ file challenge
challenge: gzip compressed data, was "dump.img", from Unix, last modified: Fri Mar 23 10:11:37 2012
```

Après décompression on obtient le fichier *dump.img*, il s'agit bien d'une image disque d'1 Go avec une partition ext2.

```
$ file dump.img
dump.img: x86 boot sector; partition 1: ID=0x83, active, starthead 1, startsector 63, 2088387
sectors, code offset 0xb8
$ parted dump.img
GNU Parted 1.8.8.1.159-1e0e
Using /mnt/hgfs/sstic2012/dump.img
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) print
Model:  (file)
Disk /mnt/hgfs/sstic2012/dump.img: 1074MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type     File system  Flags
 1      32.3kB  1069MB  1069MB  primary  ext2         boot
```

On monte la partition en précisant son offset dans le fichier, indiqué par la table des partitions. Le dossier */home/sstic* contient les fichiers du challenge.

```
$ mount -o loop,ro,offset=32256 dump.img mnt/

$ ls -la mnt/home/sstic/
total 1364
drwxr-xr-x 2 jean jean    4096 2012-03-23 09:51 .
drwxr-xr-x 3 root root    4096 2012-01-26 13:42 ..
-rw-r--r-- 1 jean jean     220 2012-01-26 13:42 .bash_logout
-rw-r--r-- 1 jean jean    3116 2012-01-26 13:42 .bashrc
-rw-r--r-- 1 root root     871 2012-03-23 09:51 irc.log
-rw-r--r-- 1 jean jean     675 2012-01-26 13:42 .profile
-rwxr-xr-x 1 root root 1048592 2012-03-23 09:29 secret
-rwxr-xr-x 1 root root     128 2012-03-23 09:30 ssticrypt

$ cat mnt/home/sstic/irc.log
#sstic-challenge: <lobster_dog> I've a problem
#sstic-challenge: <lobster_dog> lobster cat is trying to steal one of my files
#sstic-challenge: <blue_footed_booby> lobster cat ?
#sstic-challenge: <lobster_dog>
http://amazingdata.com/mediadata56/Image/2007_0921honeymoon0141_animal_fun_weird_interesting_20090803
19215810225.jpg
#sstic-challenge: <blue_footed_booby> k -_-
#sstic-challenge: <blue_footed_booby> gimme your file, I'll hide it
#sstic-challenge: <lobster_dog> k
#sstic-challenge: <blue_footed_booby> your data is secure ;) I just finished the encryption system
#sstic-challenge: <lobster_dog> ok, I secure erase it on my side!
#sstic-challenge: <blue_footed_booby> unfortunately my hard drive is making strange noises right now
... :(
-!- blue_footed_booby [~booby@galapagos] has quit [Read error: Connection reset by peer]
#sstic-challenge: <lobster_dog> ...

$ file mnt/home/sstic/ssticrypt
mnt/home/sstic/ssticrypt: ELF 32-bit MSB executable, MIPS, MIPS-I version 1 (SYSV), statically
linked, stripped
```

Le but du challenge consiste donc à analyser le programme de chiffrement crée par le fou à pieds bleus, afin de déchiffrer le fichier du chien homard !

Comme indiqué dans le log IRC, le système de fichiers est corrompu et il faut donc le réparer pour pouvoir lire correctement le binaire MIPS *ssticrypt* (qui ne fait que 128 octets avant réparation).

```
$ losetup -o 32256  /dev/loop0 dump.img

$ fsck -fy /dev/loop0
fsck from util-linux-ng 2.16
e2fsck 1.41.9 (22-Aug-2009)
Pass 1: Checking inodes, blocks, and sizes
Inode 14, i_blocks is 2064, should be 24.  Fix? yes

Inode 19, i_size is 128, should be 311296.  Fix? yes

Inode 40820, i_size is 236, should be 40960.  Fix? yes

Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 3A: Optimizing directories
Pass 4: Checking reference counts
Pass 5: Checking group summary information
Block bitmap differences:  -(26628--26882)
Fix? yes

Free blocks count wrong for group #0 (31843, counted=32098).
Fix? yes

Free blocks count wrong (232968, counted=233223).
Fix? yes


/dev/loop0: ***** FILE SYSTEM WAS MODIFIED *****
/dev/loop0: 5469/65280 files (1.5% non-contiguous), 27825/261048 blocks
```

Avant de commencer l'analyse du binaire, il est possible de démarrer le système Linux MIPS contenu dans l'image disque avec QEMU.

```
$ cp mnt/boot/vmlinux .
$ umount mnt
$ qemu-system-mips -kernel vmlinux -append "console=ttyS0 root=/dev/hda1 init=/sbin/init" -nographic
dump.img
```

Les mots de passe des comptes utilisateurs (*root* et *sstic*) peuvent être supprimés en modifiant le fichier */etc/shadow* sur la partition, ou bien cassés avec John the Ripper :

```
$ john mnt/etc/shadow
Loaded 2 password hashes with 2 different salts (FreeBSD MD5 [32/32])
pouet            (sstic)
pouet            (root)
guesses: 2  time: 0:00:14:38 (3)  c/s: 7572  trying: pouet
```

Une fois le système démarré dans QEMU, il est possible d'utiliser le programme *ssticrypt* :

```
sstic@sstic-host:~$ ./ssticrypt
--> SSTICRYPT <--
usage: ./ssticrypt [-d|-e] <key> <secure container>
        -d: uncrypt
        -e: crypt

sstic@sstic-host:~$ ./ssticrypt -d AA secret
--> SSTICRYPT <--
Error: key should be a 128-bits hexadecimal string

sstic@sstic-host:~$ ./ssticrypt -d AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA secret
--> SSTICRYPT <--
Warning: MD5 mismatch for container
Using keys AAAAAAAAAAAAAAAA / AAAAAAAAAAAAAAAA ...
Traceback (most recent call last):
  File "check.py", line 50107, in <module>
AssertionError
Error: bad key1
```

SSTICRYPT

La première étape consiste à analyser le programme *ssticrypt* (j'ai utilisé IDA). Le binaire n'est pas obfusqué et les symboles sont présents, on connait donc le nom des fonctions et des variables globales. Le code C reversé de la fonction *main* est le suivant :

```
1.  int main(int argc, char** argv)
2.  {
3.      int i = 0;
4.      char* output_filename;
5.      int encrypt_flag = 0;
6.      int bytes_read = 0;
7.      unsigned char* buf2;
8.      unsigned char* buf;
9.      int file_size;
10.     int fd;
11.     unsigned char md5_tocompare[16];
12.     unsigned char computed_hash[16];
13.     char key1[17];
14.     char key2[17];
15.     RC4_KEY rc4_key;
16.
17.     printf("--> SSTICRYPT <--\n");
18.     if(argc != 4)
19.     {
20.         printf("usage: %s [-d|-e] <key> <secure container>\n" \
21.                 "\t-d: uncrypt\n" \
22.                 "\t-e: crypt\n", argv[0]);
23.         exit(-1);
24.     }
25.     if(strlen(argv[2]) != 32)
26.     {
27.         printf("Error: key should be a 128-bits hexadecimal string\n");
28.         exit(-1);
29.     }
30.
31.     if(!strcmp(argv[1], "-e"))
32.     {
33.         encrypt_flag = 1;
34.     }
35.
36.     fd = open(argv[3], O_RDONLY);
37.     if(fd < 0)
38.     {
39.         perror("open");
40.         exit(-1);
41.     }
42.     file_size = lseek(fd, 0, SEEK_END);
43.     lseek(fd, 0, SEEK_SET);
44.     if(!encrypt_flag)
45.     {
46.         read(fd, md5_tocompare, 16);
47.         file_size -= 16;
48.     }
49.     buf = calloc(file_size, 1);
50.     if(buf == NULL)
51.     {
52.         exit(-1);
53.     }
54.     do
55.     {
56.         bytes_read += read(fd, buf, file_size);
57.     }while(bytes_read != file_size);
58.
59.     close(fd);
60.
61.     if (!encrypt_flag)
62.     {
63.         MD5(buf, file_size, computed_hash);
64.         if(memcmp(md5_tocompare, computed_hash, 16))
65.         {
66.             printf("Warning: MD5 mismatch for container\n");
67.         }
68.     }
69.
70.     buf2 = calloc(file_size, 1);
71.
72.     strncpy(key2, &argv[2][16], 16);
73.     key2[16] = '\0';
74.     strncpy(key1, argv[2], 16);
75.     key1[16] = '\0';
76.
77.     printf("Using keys %s / %s ...\n", key1, key2);
78.
79.     if(!strcmp(key1, "5132397062694567") && !strcmp(key2, "513239706269453d"))
80.     {
81.         printf("Error: You're a duck\n");
82.         exit(1);
83.     }
84.     if(!encrypt_flag)
85.     {
```

```
        check_key(key1, 1);
        check_key(key2, 2);
    }

    RC4_set_key(&rc4_key, 32, (unsigned char*) argv[2]);

    if(!encrypt_flag)
    {
        for(i=1; i < file_size; i++)
        {
            buf[i-1] ^= buf[i];
        }
    }
    RC4(&rc4_key, file_size, buf, buf2);

    if(encrypt_flag)
    {
        for(i=file_size-1; i > 0; i--)
        {
            buf2[i-1] ^= buf2[i];
        }
    }

    output_filename = calloc(strlen(argv[3]) + 5, 1);

    sprintf(output_filename, encrypt_flag ? "%s.enc" : "%s.dec", argv[3]);

    fd = open(output_filename, O_CREAT | O_WRONLY);

    if(fd < 0)
    {
        perror("open");
        exit(-1);
    }
    MD5(buf2, file_size, computed_hash);

    if(encrypt_flag)
    {
        write(fd, computed_hash, 16);
    }
    write(fd, buf2, file_size);
    close(fd);

    return 0;
}
```

La clé de 128 bits passée en paramètre (sous forme hexadécimale) est divisée en 2 parties de 64 bits, qui sont vérifiées par la fonction *check_key*. Si les 2 parties sont validées, la chaine hexadécimale est utilisée comme clé RC4 pour déchiffrer le fichier (après avoir XORé deux à deux les octets du fichier chiffré). Les 16 premiers octets du fichier chiffré correspondent à l'empreinte MD5 du reste du fichier.

Il faut donc s'intéresser à la fonction *check_key* pour tenter de retrouver la clé de chiffrement.

```
char *ram;

void check_key(char* key, int keynum)
{
    int ret;
    char cmd[132];

    if(keynum == 1)
    {
        extract_pyc();
        sprintf(cmd, "python ./check.pyc %16s", key);
        ret = system(cmd);
        unlink("./check.pyc");
        if(ret)
        {
            printf("Error: bad key1\n");
            exit(1);
        }
    }
    else
    {
        signal(SIGINT, vicpwn_quit);
        ram = malloc(0x10000);
        vicpwn_sendbuf(init_rom, init_rom_len);
        vicpwn_sendbuf(stage2_rom, stage2_rom_len);
        vicpwn_handle(key);
    }
}
```

CLÉ 1

DÉCOMPILATION PYTHON

Pour vérifier la première partie de la clé, un script Python compilé (pour Python 2.5) est extrait du binaire et exécuté. Après quelques essais infructueux pour décompiler le fichier avec des outils open-source (*unpyc*, *uncompyle*), j'ai décompilé "à la main" le script, en utilisant le module Python *dis* pour désassembler le bytecode[1].

Le script *check.pyc* contient une class *Bits* (issue du challenge SSTIC 2011[2]), une implémentation standard de l'algorithme DES ainsi qu'une implémentation en boite blanche (classe WhiteDES). Le code Python décompilé est présent en annexe.

Lors du lancement du script, le code suivant est exécuté :

```
1.   if __name__ == "__main__":
2.       WT = pickle.loads("""<instance d'objet WhiteDES sous forme de pickle>""")
3.       if len(sys.argv) == 1:
4.           print "Usage: python check.pyc <key>"
5.           print "    - key: a 64 bits hexlify-ed string"
6.           print "Example: python check.pyc 0123456789abcdef"
7.       else:
8.           K = Bits(a2b_hex(sys.argv[1]), 64)
9.           assert K[range(7,64,8)] == 175
10.          M = Bits(random.getrandbits(64), 64)
11.
12.
13.
14.          if hex(WT._cipher(M, 1)) == hex(enc(K, M)):
15.              exit(0)
16.          else:
17.              exit(1)
```

La boite blanche DES est instanciée à partir d'un objet serialisé *pickle*. Le paramètre attendu en ligne de commande est une clé DES de 64 bits (la première moitié de la clé passée à *ssticrypt*). Un bloc aléatoire est ensuite généré, et chiffré à la fois à via la boite blanche et via l'implémentation standard de DES, en utilisant la clé passée en paramètre. Si les 2 chiffrés sont différents, un code d'erreur est renvoyé : il faut donc que la clé fournie corresponde à celle "cachée" dans la boite blanche. Les bits de parité de la clé sont donnés par l'assertion *assert K[range(7,64,8)] == 175*.

DES BOITE BLANCHE

La cryptographie boite blanche à pour but de transformer un algorithme de chiffrement de manière à empêcher (ou au moins ralentir) la récupération des secrets de l'algorithme (clés) par un attaquant ayant un accès complet au système. Ces techniques sont principalement utilisées dans les systèmes de DRM, où l'utilisateur doit pouvoir déchiffrer du contenu protégé sans pouvoir facilement récupérer les clés de chiffrement.

La boite blanche DES utilisée dans le challenge correspond à celle décrite dans l'article "A White-Box DES Implementation for DRM Applications"[3] (S. Chow, P. Eisen, H. Johnson, P.C. van Oorschot), sans encodages externes (naked variant).

J'ai tout d'abord essayé d'implémenter l'attaque "Statistical Bucketing Attack on Naked Variant" présentée dans l'article, mais la description donnée était trop sommaire pour moi.

En cherchant d'autres informations, j'ai ensuite trouvé l'article "Clarifying Obfuscation: Improving the Security of White-Box Encoding"[4] (Hamilton E. Link and William D. Neumann). La section "Attack on Split T-Box Output" de cet article décrit une attaque similaire à celle présentée dans le papier original, mais avec plus de détails. En suivant "à la lettre" cet article j'ai pu implémenter une attaque fonctionnelle. Le code source (whitebox.py) est disponible en annexe.

---

[1] http://nedbatchelder.com/blog/200804/the_structure_of_pyc_files.html
[2] http://static.sstic.org/challenge2011/axel_tillequin.pdf
[3] http://crypto.stanford.edu/DRM2002/whitebox.pdf
[4] http://eprint.iacr.org/2004/025.pdf

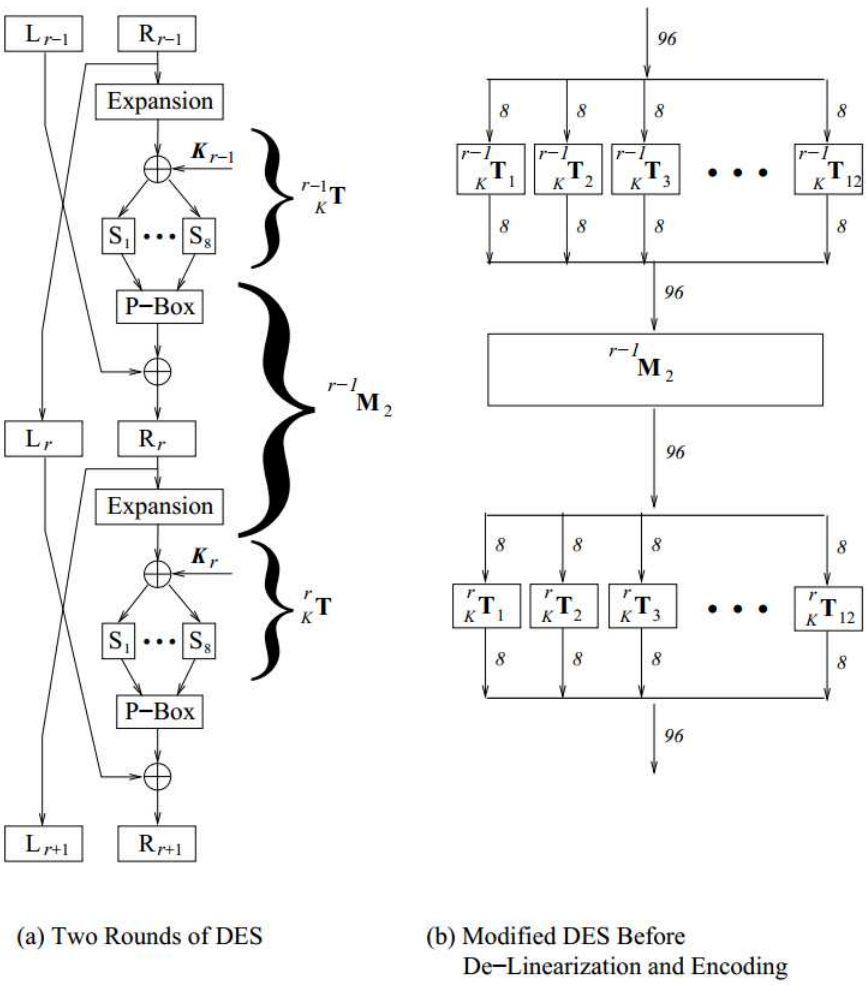


Figure 1 - Boite blanche DES

L'attaque permet de récupérer 2 sous-clés possibles pour le premier tour de l'algorithme. A partir de ces sous-clés il est possible de bruteforcer la clé en inversant le key schedule et en essayant toutes les valeurs possibles pour les 8 bits "perdus" lors de l'application de la permutation PC2. Une fois la clé trouvée, les bits de parité sont fixés aux valeurs attendues par *check.pyc*. La clé obtenue est **fd4185ff66a94afd.**

```
$ python whitebox.py
sub-subkey for sbox 0 : 101101
sub-subkey for sbox 1 : 011000
sub-subkey for sbox 2 : 111101
sub-subkey for sbox 3 : 011010
sub-subkey for sbox 3 : 110101
sub-subkey for sbox 4 : 111001
sub-subkey for sbox 5 : 011101
sub-subkey for sbox 6 : 101000
sub-subkey for sbox 7 : 100111
Found subkeys :
101101011000111101011010111001011101101000100111
101101011000111101110101111001011101101000100111
Bruteforcing last 8 bits for 2 possible subkeys
Key found \o/ 1111110101000001100001011111111101100110101010010100101011111101
Key = fd4185ff66a94afd
```

CLÉ 2

La seconde partie de la clé est vérifiée par les fonctions "*vicpwn*". La fonction *vicpwn_init* recherche un périphérique USB ayant le vendor id 0x4c1 et device id 0x9d. Une recherche Google nous indique qu'il s'agit d'une webcam 3Com HomeConnect USB Camera. Le nom des paramètres passés à la fonction *vicpwn_sendbuf* (*init_rom* et *stage2_rom*) semble indiquer que le programme *ssticrypt* charge un code binaire sur cette webcam. Il va donc falloir déterminer l'architecture CPU utilisée.

IDENTIFICATION DU FIRMWARE

Différents posts sur des mailing lists montrent des logs du module "vicam" du noyau Linux, ce qui correspond au nom "vicpwn". Quelques recherches Google plus tard, on trouve la brochure[5] du processeur d'images Vicam 3, sans doute utilisé dans la webcam 3Com. Un schéma indique que le Vicam 3 est construit autour d'un microcontrôleur 16 bits RISC (VII 16-Bit RISC MCU).

En observant les 2 blocs binaires envoyés par la fonction *vicpwn_sendbuf* et le fichier *vicam_firmware.fw* présent dans le noyau Linux, on remarque que les octets "B6 C3" sont répétés à plusieurs endroits.

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  B6 C3 31 00 02 64 E7 07 00 00 08 C0 E7 07 00 00  ¶Ã1..dç....Àç...
00000010  3E C0 E7 67 FD FF 0E C0 E7 09 DE 00 8E 00 C0 09  >Àçgýÿ.Àç.Þ.Ž.À.
00000020  40 03 C0 17 44 03 4B AF C0 07 44 03 4B AF C0 07  @.À.D.K¯À.D.K¯À.
00000030  00 00 4B AF 97 CF B6 C3 03 00 03 64 2A 00 B6 C3  ..K¯—Ï¶Ã...d*.¶Ã
00000040  01 00 06 64                                      ...d

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  B6 C3 A9 08 02 64 E7 07 76 00 AA 00 E7 07 56 F7  ¶Ã©..dç.v.ª.ç.V÷
00000010  B4 00 C8 07 1A 01 E0 07 00 40 E0 07 00 00 E0 07  ´.È...à..@à...à.
00000020  00 00 97 CF 00 00 00 00 00 00 00 00 00 00 00 00  ..—Ï............
```

La recherche de "B6 C3 opcode" sur Google fait ressortir un résultat intéressant[6] :

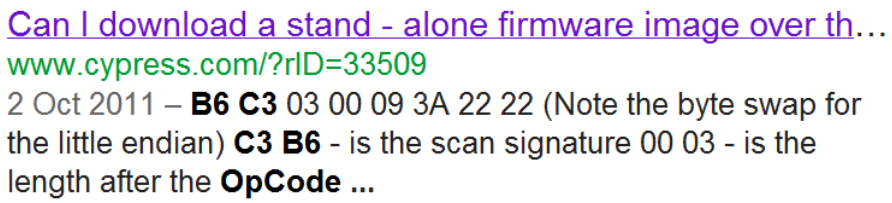


Figure 2 - Google fu

[5] http://ftp.vistaimaging.com/ViCAM-III%20Brochure.pdf

[6] http://www.cypress.com/?id=4&rID=33509

Knowledge Base Article

Can I download a stand-alone firmware image over the co-processor interface?

Last Updated: 10/02/2011

Can I download a stand-alone firmware image over the co-processor interface?

Yes! In matter of fact the CY4640 version 1.1 and greater have enabled this feature and can be used for reference. To understand how this is implemented by Cypress you first need to understand how scan signatures and scan codes are used by the BIOS. The standard process when building stand-alone code is to compile the code and then run the scanwrap or scanwrp2 programs, which embed scan signatures and scan codes in the firmware image. These scan signatures/codes can perform functions like write to a register, write to regular memory locations and start the execution of code. More information on scan signatures and scan codes can be found in the BIOS User Manual (Section 1.7.2) included in the Docs directory after the CY3663 or CY4640 software has been installed on your computer. Once a scanwrapped image is available an application can parse through this image and do the same thing that the BIOS would do with these scan codes except it can do this over the co-processor interface using LCP commands. For example one of the first things in a scan wrapped image might be to change the wait states for the external memory. If you were to look at the scan wrapped image with a HEX editor you would see:

B6 C3 03 00 09 3A 22 22 (Note the byte swap for the little endian)

C3 B6 - is the scan signature
00 03 - is the length after the OpCode
09 - is the OpCode for Write Configuration
3A - is the configuration address (adds 0xC000 because this is the configuration space)
22 22 - is the data that will be written to address 0xC03A.
So the parsing routine would recognize the scan signature as being valid and would know how long the actual data is going to be. It will then recognize that this is to write a configuration register and has the register address and data that needs to be written. Then over the co-processor interface it will use the equivalent LCP command. In this case it would equate to a COMM_WRITE_CTRL_REG LCP command. The application can then continue parsing through the scan wrapped image until it has downloaded the complete image and started execution of the code at the starting location of the firmware image.

Check the Cypress web site for updated application notes that address this topic in more detail.

Related Categories: USB Hosts

Figure 3 - Article sur le site de Cypress

A partir de cet article, on peut déterminer qu'il s'agit d'un firmware pour un contrôleur USB Cypress. Les documents suivants vont nous aider à poursuivre l'analyse :

- CY16 USB Host/Slave Controller/16-Bit RISC Processor Programmers Guide[7]
- BIOS User's Manual[8]
- USB Multi-Role Device Design By Example[9]

Un kit de développement est disponible sur le site de Cypress[10] : celui ci contient une chaine de compilation (sous Cygwin), avec en particulier une version d'*objdump* pour l'architecture CY16, qui va nous permettre de désassembler le firmware présent dans *ssticrypt*.

Avant de pouvoir désassembler le code, il est nécessaire du comprendre le format des données envoyées au périphérique. La section *BIOS Scan Operation* du document "USB Multi-Role Device Design By Example" décrit le format des *scan records*. Les 2 firmwares de *ssticrypt* utilisent les 3 opcodes SCAN suivants :

- 2 : *Request N-1 byte buffer from BIOS, set VEC to point to this buffer. Copy N-1 bytes into this buffer*
- 3 : *Add the real address pointed to by VEC to the following list of ADDR's. This is a relocation fixup.*
- 6 : *Execute VEC*

[7] http://www.cypress.com/?docID=14345
[8] http://www.cypress.com/?docID=14346
[9] http://www.cypress.com/?docID=14347
[10] http://www.cypress.com/?rID=14436

Opcode	Offset des données	Taille des données
2 : copie du payload	0x6	0x808
3 : relocations	0x8B4	0x1BC

Tableau 1 - Contenu du firmware stage2

Le firmware "init rom" contient du code d'initialisation et n'a pas été étudié en détail.

Le firmware *stage2* peut être extrait avec les commandes suivantes :

```
$ dd if=ssticrypt bs=1 skip=`echo $((0x4772c))` count=`echo $((0xa76))` > stage2
$ dd if=stage2 bs=1 skip=6 count=`echo $((0x808))` > stage2_payload
$ dd if=stage2 bs=1 skip=`echo $((0x8b4))` count=`echo $((0x1BC))` > stage2_relocs
```

Il est ensuite possible de désassembler le code avec *objdump* dans l'environnement Cypress :

```
[cy]$ cy16-elf-objdump.exe -D -b binary -m sl11r stage2_payload
```

Avant d'étudier le firmware, il faut analyser la fonction *vicpwn_handle* qui communique avec la webcam :

```
1.  char *ram; //ram = malloc(0x10000);
2.  uint32_t o = 0;
3.
4.  void vicpwn_handle(char* key)
5.  {
6.      uint8_t* layer_ptr;
7.      uint32_t layer_num = 0;
8.
9.      vicpwn_init(VICAM_DEVICE_ID);
10.     if(devCam == NULL)
11.     {
12.         printf("No Dev found.\n");
13.         exit(-1);
14.     }
15.     uint16_t* buffer = malloc(0x14);
16.     layer_ptr = layer1;
17.
18.     for(layer_num = 0; layer_num < 3; layer_num++)
19.     {
20.         load_layer(layer_ptr, layer_num, 0);
21.
22.         set_my_key(key, layer_num, 0xA000);
23.
24.         while(1)
25.         {
26.             if (ram[0x8000] != 0x0 && o == 0)
27.                 break;
28.
29.             o &= 0xFFFFFFF0;
30.             if((o & 0xFFF0) != 0xFFF0)
31.             {
32.                 //send 16 bytes of ram to device
33.                 memcpy(buffer, &ram[o], 0x10);
34.                 //usb_control_msg(dev, requesttype, request, value, index, bytes, size, timeout);
35.                 //0x40 = Host to device , vendor
36.                 usb_control_msg(devCam, 0x40, 0x51, o, 1, (char*) buffer, 0x10, 1000);
37.             }
38.             memset(buffer, 0x0, 0x14);
39.
40.             //read 20 bytes from device
41.             //usb_control_msg(dev, requesttype, request, value, index, bytes, size, timeout);
42.             //0xC0 = Device to host, vendor
43.             uint32_t bytesRead = usb_control_msg(devCam, 0xC0, 0x56, 0x0, 0x0, (char*) buffer, 0x14, 1000);
44.
45.             //format : old page data(16 bytes) | old_page | new_page
46.             uint16_t evicted_page = swap_word(ntohs(buffer[8]) & 0xFFFF);
47.             o = swap_word(ntohs(buffer[9]) & 0xFFFF);
48.
49.             bytesRead = 0x14;
50.
51.             //if evicted_page entry was used and dirty write back to ram
52.             if(((evicted_page & 0xFFF0) != 0xFFF0) && ((evicted_page & 1) != 0))
53.             {
54.                 evicted_page &= 0xFFFFFFF0;
55.                 memcpy(&ram[evicted_page], buffer, 0x10);
56.             }
57.         }
```

```
58.          layer_ptr = vicpwn_check(layer_num, 0xA000, key);
59.          ram[0x8000] = 0x0;
60.      }
61.
62.      usb_release_interface(devCam, 0);
63.      free(buffer);
64.      vicpwn_close(devCam);
65.  }
```

La boucle dans la fonction *vicpwn_handle* communique avec la webcam via des requêtes USB ayant pour code 0x50 et 0x56, et transfère des données vers et depuis le tableau *ram*.

Les noms des variables globales *ram* (tableau de 64Ko) et *layers* laissent penser que le code exécuté sur la webcam serait en fait une machine virtuelle, dont la mémoire « RAM » est située sur la machine hôte. En reversant la partie CY16 on se rend compte que c'est effectivement le cas, et que les auteurs du challenge ont implémenté une sorte de MMU over USB (Figure 5).

La fonction *load_layer* charge un bloc binaire (code pour la machine virtuelle) dans le tableau *ram*. Pour chaque layer exécuté, la fonction *set_my_key* initialise les données d'entrée dans la mémoire de la VM.

```
1.  void set_my_key(char* key, uint32_t layer_num, uint32_t offset)
2.  {
3.      char local_key[0x100];
4.      strcpy(local_key, key);
5.
6.      local_key[(layer_num+2)*4] = '\0';
7.
8.      uint32_t var_14 = htonl(strtoul(&local_key[layer_num*4], NULL, 0x10));
9.      //0xAABBCCDD => 0xBBAADDCC
10.     swap_key(&var_14);
11.
12.     memcpy(&ram[offset], &var_14, 4);
13.
14.     if(layer_num == 1)
15.     {
16.         memcpy(&ram[0x10+offset], blob, 0x100);
17.     }
18.     else if (layer_num == 2)
19.     {
20.         memcpy(&ram[0x10+offset], blah, 0x20);
21.     }
22. }
```

Les 3 layers vont donc traiter 3 parties de la clé passée en paramètre :

	[0xA000]	[0xA002]
Layer 1	0x1122	0x3344
Layer 2	0x3344	0x5566
Layer 3	0x5566	0x7788

Figure 4 - Entrées des différents layers pour la clé 1122334455667788

L'analyse du firmware CY16 a été faite en statique sur la sortie d'objdump, en réécrivant un pseudo code C correspondant au fur et à mesure.

L'extrait suivant montre l'initialisation du vecteur d'interruption *SUSB1_VENDOR_INT* : la routine de traitement de cette interruption compare la valeur de la requête USB avec les valeurs 0x51 et 0x56, ce qui correspond bien aux valeurs présentes dans ssticrypt. Lors de la réception de données depuis l'hôte, le callback appelle la fonction à l'adresse 0x4da, qui correspond à l'interpréteur de la machine virtuelle.

```
stage2_payload:     file format binary

Disassembly of section .data:

00000000 <.data>:
   0:   e7 07 76 00    mov w[0xaa],0x76             ;SUSB1_VENDOR_INT
   4:   aa 00
   6:   e7 07 56 f7    mov w[0xb4],0xfffff756       ;device descriptor?
   a:   b4 00
   c:   c8 07 1a 01    mov r8,0x11a
  10:   e0 07 00 40    mov w[r8++],0x4000
  14:   e0 07 00 00    mov w[r8++],0x0
  18:   e0 07 00 00    mov w[r8++],0x0
  1c:   97 cf          ret
        ...
  52:   00 00          mov r0,r0
  54:   f1 ff          *unknown*
  56:   f2 ff          *unknown*
  58:   f3 ff          *unknown*
  5a:   f4 ff          *unknown*
  5c:   f5 ff          *unknown*
  5e:   ff ff          *unknown*
  60:   ff ff          *unknown*
  62:   ff ff          *unknown*
  64:   ff ff          *unknown*
  66:   ff ff          *unknown*
  68:   00 00          mov r0,r0                    ; &ram_cache[0][0]
  6a:   54 00          mov w[r12],r1                ; &page_table[0]
  6c:   00 00          mov r0,r0
  6e:   00 00          mov r0,r0
  70:   00 00          mov r0,r0
  72:   00 00          mov r0,r0
  74:   00 00          mov r0,r0

usb_interrupt_handler:
  76:   00 0e 01 00    mov r0,b[r8+0x1]             ; r0 = bRequest
  7a:   c0 57 51 00    cmp r0,0x51
  7e:   05 c0          jzs 0x8a
  80:   c0 57 56 00    cmp r0,0x56
  84:   14 c0          jzs 0xae
  86:   9f cf e8 00    jmpl 0xe8
read_from_host_51:
  8a:   c9 07 68 00    mov r9,0x68
  8e:   41 08          mov r1,w[r9++]               ; r1 = next_pagein_ptr
  90:   4a 08          mov r10,w[r9++]              ; r10 = next_pagein_pte
  92:   48 d8          addi r8,0x2
  94:   22 08          mov w[r10++],w[r8++ ]        ; next_pagein_pte = wValue = o
  96:   21 08          mov w[r9++],w[r8++]          ; wIndex = 1
  98:   48 02          mov r8,r9
  9a:   61 94          xor w[r9++],w[r9]            ; next_link = 0
  9c:   61 00          mov w[r9++],r1               ; addr = next_pagein_ptr
  9e:   e1 07 10 00    mov w[r9++],0x10             ; length = 16
  a2:   e1 07 f6 00    mov w[r9++],0xf6             ; recv_callback
  a6:   c1 07 00 80    mov r1,0xffff8000            ; endpoint0
  aa:   51 af          int 0x51                     ; int 81d : SUSB1_RECEIVE_INT
  ac:   97 cf          ret

send_to_host_56:
  ae:   c9 07 68 00    mov r9,0x68
  b2:   41 08          mov r1,w[r9++]               ; r1 = next_pagein_ptr
  b4:   4c 00          mov r12,r1
  b6:   cc 17 10 00    add r12,0x10
  ba:   ca 07 20 01    mov r10,0x120
  be:   22 03          mov w[r10++],r12
  c0:   22 05          mov w[r10++],w[r12]
  c2:   22 0d 02 00    mov w[r10++],w[r12+0x2]
  c6:   48 08          mov r8,w[r9++]
  c8:   c3 07 14 00    mov r3,0x14
  cc:   24 04          mov w[r12++],w[r8]
  ce:   c9 07 6e 00    mov r9,0x6e
  d2:   54 04          mov w[r12],w[r9]
  d4:   48 02          mov r8,r9
  d6:   61 94          xor w[r9++],w[r9]            ; next_link = 0
  d8:   61 00          mov w[r9++],r1               ; address = next_pagein_ptr
  da:   e1 00          mov w[r9++],r3               ; length = 20 (16+2+2)
  dc:   e1 07 e8 00    mov w[r9++],0xe8             ; send_callback
  e0:   c1 07 00 80    mov r1,0xffff8000
  e4:   50 af          int 0x50                     ; int 80d : SUSB1_SEND_INT
  e6:   97 cf          ret
```

```
send_callback:
  e8:   59 af          int 0x59                    ; int 89d SUSB1_FINISH_INT
  ea:   ca 07 20 01    mov r10,0x120
  ee:   8c 08          mov r12,w[r10++]
  f0:   a4 08          mov w[r12++],w[r10++]
  f2:   a4 08          mov w[r12++],w[r10++]
  f4:   97 cf          ret

recv_callback:
  f6:   9f af da 04    call 0x4da                  ; vm_interpreter
  fa:   59 af          int 0x59                    ; int 89d SUSB1_FINISH_INT
  fc:   97 cf          ret
```

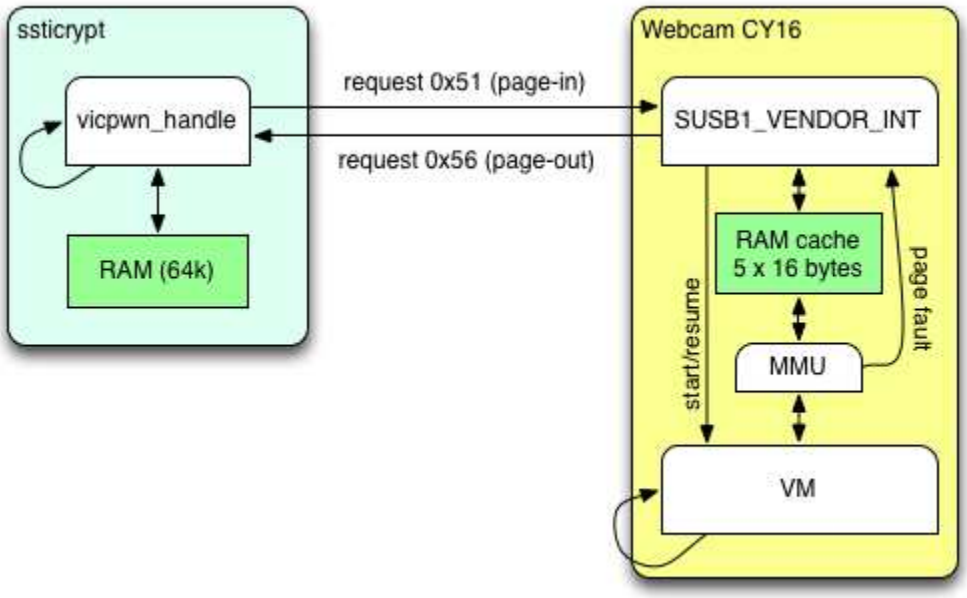


Figure 5 – Fonctionnement de la machine virtuelle et MMU/USB

Les fonctions présentes dans le firmware CY16 sont resumées dans le tableau suivant :

Adresse (relative)	Description
0x76	Gestionnaire d'interruption SUSB1_VENDOR_INT
0x146	Shift right (return r0 >> r1)
0x156	Shift left (return r0 << r1)
0x162	Lecture mémoire VM
0x166	Ecriture mémoire VM
0x1da	MMU : translation adresse RAM VM vers adresse cache RAM
0x268	Lit r0 bits à partir du pointeur d'instruction
0x346	Copie l'opérande à l'adresse 0x126 vers l'adresse 0x134
0x35e	Affecte le résultat d'une instruction dans l'opérande de destination
0x3c4	Lit un opérande et le stocke a l'adresse 0x126
0x4da	Interpréteur de la VM
0x814	Lecture mémoire VM (type d'opérande 3)
0x848	Ecriture mémoire VM (type d'opérande 3)

Tableau 2 - Fonctions du firmware CY16

Adresse (relative)	Description
0x0	uint8_t ram_cache[5][16]
0x54	uint16_t page_table[5];
0x5e	uint16_t page_age[5];
0x68	uint8_t* next_pagein_ptr; uint16_t* next_pagein_pte; uint16_t index; //usb message index (1)
0x6e	typedef struct USB_INT_PARAM { uint16* next_link; uint16 addr; uint16 length; uint16 call_back; } USB_INT_PARAM;
0xfe	uint16_t vm_registers[14];

0x11a	struct vm_ctx
	{
	uint16_t stack_ptr; //= 0x4000
	uint16_t pc_byte; //=0x0
	uint16_t pc_bit; //=0x0
	} ;

0x120	Buffer temporaire
0x126	Opérande 1
0x134	Opérande 2
0x144	Registres de flags

Tableau 3 - Variables et structures du firmware CY16

MACHINE VIRTUELLE

La machine virtuelle implémentée dans le firmware CY16 utilise des instructions encodées sur un nombre variable de bits. La fonction à l'adresse 0x268 est utilisée pour lire n bits à partir du pointeur d'instruction courant.

Bits	0	1-9	10-13	14	15-n
Description	Flags_register	condition	opcode	Set flags	Varie selon l'opcode

Figure 6 - Format des instructions

Les instructions sont conditionnelles, et de plus la machine virtuelle possède 2 registres de flags : le premier bit de l'instruction indique lequel utiliser. Les conditions sont codées sur 8 bits, mais en pratique seuls 4 bits sont utilisés, selon le registre de flags choisi. Ces 4 bits de condition correspondent à ceux de l'architecture CY16, et sont gérés dans l'interpréteur par du code auto modifiant qui patche une instruction de saut conditionnel. Le bit 14 indique si le registre de flags doit être mis à jour une fois l'instruction exécutée.

Les différents opcodes supportés sont listés dans le tableau suivant :

Opcode	Description
0	AND
1	OR
2	NOT
3	SHIFT
4	MOV
5	?
6	CALL
7	JMP

Tableau 4 - Instructions de la machine virtuelle

Bits	0	1-2	3 – n
Description	Taille 0 : 8 bits 1 : 16 bits	Type	Varie selon le type

Tableau 5 - Format des operandes

DÉCOMPILATION SEMI-AUTOMATIQUE

Une fois le format des instructions identifié, j'ai écrit un désassembleur (*sstic_vm.py* en annexe). Après quelques ajustements, j'ai pu désassembler le layer1, et vérifier la cohérence du résultat : le programme commence par lire les valeurs aux adresses *0xA000* et *0xA002* (morceau de clé à verifier), et se termine en écrivant *0xdeadbeef* à l'adresse *0x8000*. Enfin, un saut vers l'adresse 0 est effectué, ce qui déclenche la condition d'arrêt de la boucle dans *vicpwn_handle*. La suite d'instructions « *mov r0, 0xaa* » avant le saut sert à forcer la MMU à renvoyer les dernières pages mémoire modifiées au programme ssticrypt.

```
0000:0      mov    r12, [0xa000]
0005:1      mov    r3, [0xa002]
000a:2      mov    r0, r12
000d:5      shr    r0, 8

...
0524:4      movZ0  [0xa000], [0xc1a4]
052b:3      movZ0  [0xdfd0], r6
0530:4      movZ0  [0xe738], 0xbeef
0537:1      notZ0  [0xe738]
053b:3      andZ0  [0xdfd0], [0xe738]
0542:2      movZ0  [0xd25c], r6
0547:3      notZ0  [0xd25c]
```

```
054b:5      andZ0  [0xd25c], 0xbeef
0552:2      orZ0   [0xdfd0], [0xd25c]
0559:1      movZ0  [0xa002], [0xdfd0]
0560:0      mov    [0xcb1e], r1
0565:1      or     [0xcb1e], r1
056a:2      mov    [0x8000], 0xdead
0570:7      mov    [0x8002], 0xbeef
0577:4      mov    r0, 0xaa
057c:3      mov    r0, 0xaa
...
064e:0      mov    r0, 0xaa
0652:7      j      0:0
```

Plutôt que d'essayer de reverser le code désassemblé, j'ai choisi de modifier le désassembleur pour générer du code C. Etant donné que les instructions sont simples et peu nombreuses, il est possible de décompiler le code sans trop de problèmes : chaque instruction sera représentée par une ligne de code C. L'idée est de pouvoir appeler en boucle la fonction décompilée pour bruteforcer la valeur correcte de la clé. Durant le challenge, je n'ai pas implémenté la décompilation des sauts et instructions conditionnelles : j'ai donc du corriger manuellement la sortie du "décompilateur" pour obtenir un code correct, d'où l'appellation semi-automatique :-).

La vérification faite dans *vicpwn_check* après l'éxécution du layer 1 consiste à tester si la valeur à l'adresse *0xA000* est différente de celle placée initialement par *set_my_key*. Cette adresse est modifiée de façon conditionnelle à un seul endroit dans le code du layer 1 (aux adresses *0524:4* et *0559:1*), il suffit donc de placer la condition d'arrêt de notre bruteforce à cet endroit.

```
$ ./layer1
Output: 0x8cfa 0xd5fb
Key? 0xe5df 0x94e3
```

Le second layer est déchiffré par la fonction *vicpwn_check* :

```
1.  else if(layer_num == 0)
2.  {
3.      uint32_t* output32 = (uint32_t*) output;
4.
5.      output32[0] = ((uint32_t*)layer_ptr)[0] ^ output_key;
6.
7.      for(i=1; i < layer_length/4; i++)
8.      {
9.          output[i] = ((uint32_t*)layer_ptr)[i] ^ output32[i-1];
10.     }
11.     if( output_key == key_int_swapped)
12.     {
13.         printf("Error: bad key2\n");
14.         exit(1);
15.     }
16. }
```

On peut donc déchiffrer le layer 2 avec la clé *0x8cfad5fb* (en big endian, voir *decrypt_layers.py* en annexe).

L'entrée pour le layer 2 est constituée des octets 3 à 6 de la clé (les octets 3 et 4 sont connus par le bruteforce du layer 1), ainsi qu'un tableau de 256 octets (variable globale *blob*). Le résultat attendu par *vicpwn_check* est une permutation RC4, utilisée pour déchiffrer le dernier layer :

```
1.  if(layer_num == 1)
2.  {
3.      //output of layer 2 is an RC4 state (256 bytes) used to decrypt layer 3
4.      memcpy(layer2_rc4_state, &ram[0x10+offset], 0x100);
5.
6.      if(layer2_rc4_state[0xFE] == 0xFF && layer2_rc4_state[0xFF] == 0xFF)
7.      {
8.          printf("Error: bad key2\n");
9.          exit(1);
10.     }
11.
12.     for(i=0, idx2=0; i < layer_length; i++)
13.     {
14.         idx2 = (idx2 + 1) % 256;
15.
16.         var_143 += layer2_rc4_state[idx2];
17.         var_144 = layer2_rc4_state[idx2];
```

```
        layer2_rc4_state[idx2] = layer2_rc4_state[var_143];
        layer2_rc4_state[var_143] = var_144;

        uint8_t keyStreamByte = layer2_rc4_state[(layer2_rc4_state[idx2] + layer2_rc4_state[var_143]) & 0xFF] & 0xFF;

        output[i] = layer_ptr[i] ^ keyStreamByte;
    }
}
```

En bruteforcant les octets 5 et 6, on obtient la permutation RC4 suivante, qui permet de déchiffrer le dernier layer (voir *decrypt_layers.py*).

```
layer3box = [   0xE3, 0xEF, 0x1D, 0x26, 0xC2, 0x00, 0x44, 0x2E, 0xB3, 0xF0, 0x10, 0x7E,
                0xFC, 0x6B, 0x06, 0xD0, 0x31, 0xE4, 0x3E, 0x6D, 0x8F, 0x9D, 0xB0, 0x2A,
                0xF6, 0x43, 0x7D, 0x53, 0x58, 0x15, 0x11, 0x19, 0x02, 0xDB, 0xA0, 0x8E,
                0x78, 0x25, 0x04, 0x5B, 0xA4, 0xA2, 0x13, 0xD4, 0x0A, 0xBC, 0x6E, 0xD3,
                0x8D, 0x85, 0xCD, 0x84, 0x81, 0x87, 0xA3, 0x3C, 0x57, 0xC0, 0xDC, 0xA6,
                0x73, 0xEE, 0x17, 0x0D, 0x61, 0xAE, 0x36, 0x83, 0x8B, 0x7C, 0xA8, 0x51,
                0xF2, 0xCE, 0x59, 0xFD, 0x91, 0x6A, 0x1A, 0x4F, 0x5A, 0x4E, 0x70, 0xF1,
                0xAC, 0x98, 0x2F, 0xD6, 0x7B, 0x49, 0x76, 0xB9, 0x0E, 0xB4, 0xF8, 0x8A,
                0x5C, 0x7A, 0xB8, 0x9E, 0xDA, 0xD8, 0x23, 0xED, 0xDD, 0x6C, 0x0B, 0x29,
                0x3F, 0x64, 0x62, 0xFF, 0xE7, 0xD2, 0x2B, 0x77, 0xD7, 0x94, 0x48, 0x39,
                0x66, 0xA9, 0x4D, 0x35, 0x8C, 0xB5, 0x28, 0x1C, 0x60, 0x9F, 0x37, 0x52,
                0x63, 0xBD, 0x99, 0x75, 0x1E, 0x34, 0xFA, 0x5F, 0x09, 0x67, 0x16, 0x3A,
                0xD9, 0x71, 0xF4, 0x80, 0xC6, 0xE0, 0x89, 0x79, 0x9A, 0xEB, 0x9B, 0x90,
                0x46, 0xFE, 0xC5, 0xBF, 0x21, 0x6F, 0xE2, 0xF5, 0x56, 0xD1, 0xCF, 0x74,
                0xB1, 0xA5, 0xE1, 0x47, 0xD5, 0xE6, 0x27, 0x86, 0x3D, 0x1F, 0xCA, 0xC1,
                0x1B, 0x05, 0xB6, 0xCC, 0xAA, 0x4B, 0x69, 0xE9, 0x03, 0xC8, 0xEA, 0x12,
                0xC4, 0xC7, 0x96, 0xCB, 0xC9, 0xE8, 0x55, 0x82, 0x95, 0xF7, 0x07, 0x20,
                0x08, 0x0C, 0xBB, 0x5E, 0x93, 0xAB, 0x14, 0x3B, 0xBA, 0x97, 0x4A, 0x33,
                0x7F, 0x54, 0x22, 0x9C, 0xDF, 0xDE, 0x42, 0x40, 0x0F, 0x24, 0xB7, 0x50,
                0x30, 0x2C, 0xAD, 0x01, 0xFB, 0xF9, 0x32, 0x5D, 0x38, 0xAF, 0xC3, 0x18,
                0x72, 0xE5, 0xF3, 0x45, 0x88, 0x2D, 0x65, 0xA7, 0x4C, 0xEC, 0x68, 0x41,
                0xB2, 0xA1, 0x92, 0xBE
                ]
```

De la même manière pour le dernier layer, on bruteforce les 2 derniers octets en comparant la valeur de sortie à l'adresse **0xA010** à la chaine "Woot !! Smells good :)" encodée en base 64.

```
else if(layer_num == 2)
{
    //"Woot !! Smells good :)"
    if(strncmp(&ram[0x10+offset] , "V29vdCAhISBTbWVsbHMgZ29vZCA6KQ==", 0x20))
    {
        printf("Error: bad key2\n");
        exit(1);
    }
    free(output);
    output = NULL;
}
```

```
$ ./layer1
Output: 0x8cfa 0xd5fb
Key? 0xe5df 0x94e3
$ ./layer2
Creating file blob_94e3_f63d.bin
Key? 0x94e3 0xf63d
$ ./layer3
Key? 0xf63d 0x8937
V29vdCAhISBTbWVsbHMgZ29vZCA6KQ==
```

La seconde moitié de la clé est donc **e5df94e3f63d8937**.

CYPRESS DU BUT

Une fois la clé trouvée, il ne reste plus qu'à déchiffrer le fichier *secret*. Seul problème, même après avoir réparé la partition avec *fsck*, le fichier est corrompu : lors du lancement de ssticrypt, la vérification du MD5 échoue et le fichier déchiffré semble valide mais tronqué.

```
$ losetup -o 32256  /dev/loop0 dump.img
$ debugfs /dev/loop0
debugfs 1.41.9 (22-Aug-2009)
debugfs:  show_inode_info /home/sstic/secret
Inode: 14   Type: regular    Mode:  0755   Flags: 0x0
Generation: 163417969    Version: 0x00000000
User:     0   Group:     0   Size: 1048592
File ACL: 0    Directory ACL: 0
Links: 1   Blockcount: 24
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x4f6c3483 -- Fri Mar 23 09:29:55 2012
atime: 0x4f7f4cd1 -- Fri Apr  6 22:06:41 2012
mtime: 0x4f6c3483 -- Fri Mar 23 09:29:55 2012
Size of extra inode fields: 0
BLOCKS:
(0-2):26625-26627
TOTAL: 3
```

A l'aide de l'outil *debugfs*, le problème est identifié : la taille du fichier est 1 Mo, mais seuls 3 blocs (de 1 ko chacun) sont référencés dans l'inode. En observant la zone correspondant à ces blocs avec un éditeur hexadécimal, on remarque que les blocs suivants contiennent également des données. Pour récupérer le fichier complet, il suffit en fait d'extraire les 1048592 octets consécutifs à partir du début du premier bloc du fichier, en omettant le 13ième bloc qui est un bloc indirect : ce bloc pointe vers les suivants, car seuls les 12 premiers blocs peuvent être référencés dans l'inode[11].

Les commandes suivantes permettent d'extraire le fichier *secret* :

```
$ dd if=dump.img bs=1 skip=`echo $((0x6808E00))` count=`echo $((0xC000))` > secret
$ dd if=dump.img bs=1 skip=`echo $((0x6815E00))` count=`echo $((0xF4010))` >> secret
```

[11] http://en.wikipedia.org/wiki/Ext2#Inodes

CONCLUSION

Une fois le fichier *secret* extrait, on peut enfin le déchiffrer avec la clé :

```
$ python ssticdecrypt.py secret fd4185ff66a94afde5df94e3f63d8937
MD5 check OK
Writing file secret.dec

$ file secret.dec
secret.dec: Linux rev 1.0 ext2 filesystem data, UUID=ace27cef-09d4-4d79-ad64-42597535b42e

$ mount -o loop secret.dec mnt/
$ ls mnt/
lobster  lost+found
$ file mnt/lobster
mnt/lobster: RIFF (little-endian) data, AVI, 352 x 264, ~15 fps, video: H.264 X.264 or H.264, audio:
MPEG-1 Layer 3 (stereo, 44100 Hz)
```

Le fichier *lobster* est une vidéo qui donne l'adresse email de validation du challenge :

Figure 7 - Le fameux chien homard

Merci aux organisateurs pour ce challenge original !

RÉFÉRENCES

- The structure of .pyc files, Ned Batchelder
  - http://nedbatchelder.com/blog/200804/the_structure_of_pyc_files.html
- Solution du Challenge SSTIC 2011, Axel Tillequin
  - http://static.sstic.org/challenge2011/axel_tillequin.pdf
- A White-Box DES Implementation for DRM Applications (S. Chow, P. Eisen, H. Johnson, P.C. van Oorschot)
  - http://crypto.stanford.edu/DRM2002/whitebox.pdf
- Clarifying Obfuscation: Improving the Security of White-Box Encoding (Hamilton E. Link and William D. Neumann)
  - http://eprint.iacr.org/2004/025.pdf
- Digital Imaging Engine Single Chip Imaging Processor
  - http://ftp.vistaimaging.com/ViCAM-III%20Brochure.pdf
- Knowledge Base Article - Can I download a stand-alone firmware image over the co-processor interface?
  - http://www.cypress.com/?id=4&rID=33509
- CY16 USB Host/Slave Controller/16-Bit RISC Processor Programmers Guide
  - http://www.cypress.com/?docID=14345
- BIOS User's Manual
  - http://www.cypress.com/?docID=14346
- USB Multi-Role Device Design By Example
  - http://www.cypress.com/?docID=14347
- CY3663 CD-ROM Image v1.0
  - http://www.cypress.com/?rID=14436
- Wikipedia
  - http://en.wikipedia.org/wiki/Data_Encryption_Standard
  - http://en.wikipedia.org/wiki/Ext2#Inodes
  - http://en.wikipedia.org/wiki/Blue-footed_Booby
- http://www.youtube.com/watch?v=A3uF1-I5uzA
- http://knowyourmeme.com/memes/dog-fort

ANNEXE – CODES SOURCES

Les codes sources présentés sont inclus dans le fichier *sstic2012.tgz* attaché au PDF.

SSTICRYPT.C

```
1.  //gcc -lusb -lcrypto -Wall ssticrypt.c
2.  #include <stdio.h>
3.  #include <string.h>
4.  #include <stdlib.h>
5.  #include <stdint.h>
6.  #include <signal.h>
7.  #include <sys/types.h>
8.  #include <sys/stat.h>
9.  #include <fcntl.h>
10. #include <usb.h>
11. #include <arpa/inet.h>
12. #include <openssl/md5.h>
13. #include <openssl/rc4.h>
14. #include "ssticrypt.h"
15.
16. #define VICAM_DEVICE_ID 0x9D
17. #define VICAM_VENDOR_ID 0x4C1
18.
19. struct usb_dev_handle* vicpwn_init(uint32_t deviceId);
20. void vicpwn_quit(int sig);
21. void vicpwn_close(struct usb_dev_handle*);
22. void vicpwn_sendbuf(char* buf, uint32_t len);
23.
24. void* vicpwn_check(uint32_t layer_num, uint32_t offset, char* key);
25.
26. void check_key(char* key, int keynum);
27. void vicpwn_handle(char* key);
28.
29. struct usb_dev_handle* devCam = NULL;
30.
31. void extract_pyc()
32. {
33.     int fd = open("./check.pyc", O_CREAT | O_WRONLY);
34.
35.     if(fd < 0)
36.     {
37.         perror("open");
38.         exit(1);
39.     }
40.
41.     //write(fd, check_pyc, check_pyc_len);
42.     close(fd);
43. }
44.
45. char *ram;
46.
47. void check_key(char* key, int keynum)
48. {
49.     int ret;
50.     char cmd[132];
51.
52.     if(keynum == 1)
53.     {
54.         //extract_pyc();
55.         sprintf(cmd, "python ./check.pyc %16s", key);
56.         ret = system(cmd);
57.         unlink("./check.pyc");
58.         if(ret)
59.         {
60.             printf("Error: bad key1\n");
61.             exit(1);
62.         }
63.     }
64.     else
65.     {
66.         signal(SIGINT, vicpwn_quit);
67.         ram = malloc(0x10000);
68.         //vicpwn_sendbuf(init_rom, init_rom_len);
69.         //vicpwn_sendbuf(stage2_rom, stage2_rom_len);
70.         vicpwn_handle(key);
71.     }
72. }
73.
74. //0xAABBCCDD => 0xBBAADDCC
75. void swap_key(uint32_t* key)
76. {
77.     uint32_t value = *key;
```

```
78.
79.     uint8_t a = value >> 24;
80.     uint8_t b = value >> 16;
81.     uint8_t c = value >> 8;
82.     uint8_t d = value & 0xFF;
83.
84.     uint32_t result = (a << 16) | (b << 24) | (d << 8) | c;
85.
86.     *key = result;
87. }
88.
89. struct usb_dev_handle* vicpwn_init(uint32_t deviceId)
90. {
91.     /*
92.     usb_init();
93.     usb_find_busses();
94.     usb_find_devices();
95.     struct usb_bus* busses = usb_get_busses();
96.
97.     while(busses != NULL)
98.     {
99.     vendor == VICAM_VENDOR_ID
100.    devCam = usb_open( );
101.    }
102.
103.    */
104.    return devCam;
105. }
106.
107. void vicpwn_close(struct usb_dev_handle* dev)
108. {
109.    usb_close(devCam);
110. }
111.
112. void vicpwn_sendbuf(char* buf, uint32_t len)
113. {
114.     vicpwn_init(VICAM_DEVICE_ID);
115.     usb_control_msg(devCam, 0x40, 0xFF, 0x00, 0x0, buf, len, 1000);
116.
117.     vicpwn_close(devCam);
118. }
119.
120. void vicpwn_quit(int sig)
121. {
122.     vicpwn_close(0);//wut
123.     exit(0);
124. }
125.
126.
127. void set_my_key(char* key, uint32_t layer_num, uint32_t offset)
128. {
129.     char local_key[0x100];
130.     strcpy(local_key, key);
131.
132.     local_key[(layer_num+2)*4] = '\0';
133.
134.     uint32_t var_14 = htonl(strtoul(&local_key[layer_num*4], NULL, 0x10));
135.     swap_key(&var_14);
136.
137.     memcpy(&ram[offset], &var_14, 4);
138.
139.     if(layer_num == 1)
140.     {
141.         memcpy(&ram[0x10+offset], blob, 0x100);
142.     }
143.     else if (layer_num == 2)
144.     {
145.         memcpy(&ram[0x10+offset], blah, 0x20);
146.     }
147. }
148.
149. void load_layer(uint8_t* layer, uint32_t layer_num, uint32_t offset)
150. {
151.     memcpy(layer, &ram[offset],layers_len[layer_num]);
152.
153.     if(layer_num > 0)
154.     {
155.         free(layer);
156.     }
157. }
158.
159. int swap_word(uint16_t p)
160. {
161.     uint8_t varD = p & 0xFF;
162.     uint8_t varE = (p >> 8) & 0xFF;
163.
164.     uint32_t z = (((varD << 8) << 16 ) >> 16) | varE;
165.
```

```
    uint32_t var10 = (z << 16) >> 16;

    return var10;
}

uint32_t o = 0;

void vicpwn_handle(char* key)
{
    uint8_t* layer_ptr;
    uint32_t layer_num = 0;

    vicpwn_init(VICAM_DEVICE_ID);
    if(devCam == NULL)
    {
        printf("No Dev found.\n");
        exit(-1);
    }
    uint16_t* buffer = malloc(0x14);
    layer_ptr = layer1;

    for(layer_num = 0; layer_num < 3; layer_num++)
    {
        load_layer(layer_ptr, layer_num, 0);

        set_my_key(key, layer_num, 0xA000);

        while(1)
        {
            if (ram[0x8000] != 0x0 && o == 0)
                break;

            o &= 0xFFFFFFF0;
            if((o & 0xFFF0) != 0xFFF0)
            {
                //send 16 bytes of ram to device
                memcpy(buffer, &ram[o], 0x10);
                //usb_control_msg(dev, requesttype, request, value, index, bytes, size, timeout);
                //0x40 = Host to device , vendor
                usb_control_msg(devCam, 0x40, 0x51, o, 1, (char*) buffer, 0x10, 1000);
            }
            memset(buffer, 0x0, 0x14);

            //read 20 bytes from device
            //usb_control_msg(dev, requesttype, request, value, index, bytes, size, timeout);
            //0xC0 = Device to host, vendor
            uint32_t bytesRead = usb_control_msg(devCam, 0xC0, 0x56, 0x0, 0x0, (char*) buffer, 0x14, 1000);

            //format : old page data(16 bytes) | old_page | new_page
            uint16_t evicted_page = swap_word(ntohs(buffer[8]) & 0xFFFF);
            o = swap_word(ntohs(buffer[9]) & 0xFFFF);

            bytesRead = 0x14;

            //if evicted_page entry was used and dirty write back to ram
            if(((evicted_page & 0xFFF0) != 0xFFF0) && ((evicted_page & 1) != 0))
            {
                evicted_page &= 0xFFFFFFF0;
                memcpy(&ram[evicted_page], buffer, 0x10);
            }
        }
        layer_ptr = vicpwn_check(layer_num, 0xA000, key);
        ram[0x8000] = 0x0;
    }

    usb_release_interface(devCam, 0);
    free(buffer);
    vicpwn_close(devCam);
}


//check key and decrypt next layer
void* vicpwn_check(uint32_t layer_num, uint32_t offset, char* key)
{
    uint32_t output_key;
    char local_key[16];
    char layer2_rc4_state[0x100];
    uint32_t i,j,idx2;
    uint32_t var_143 = 0;
    uint32_t var_144 = 0;

    char* layer_ptr = all_layers[layer_num+1];
    uint32_t layer_length = layers_len[layer_num+1];
    char* output = calloc(1, layer_length);

    memcpy(&output_key, &ram[offset], 4);

    swap_key(&output_key);
```

```
254.
255.     strncpy(local_key, key, 16);
256.     local_key[8] = '\0';
257.
258.     uint32_t key_int_swapped = htonl(strtoul(local_key, NULL, 16));
259.
260.     if(layer_num == 1)
261.     {
262.         //output of layer 2 is an RC4 state (256 bytes) used to decrypt layer 3
263.         memcpy(layer2_rc4_state, &ram[0x10+offset], 0x100);
264.
265.         if(layer2_rc4_state[0xFE] == 0xFF && layer2_rc4_state[0xFF] == 0xFF)
266.         {
267.             printf("Error: bad key2\n");
268.             exit(1);
269.         }
270.
271.         for(i=0, idx2=0; i < layer_length; i++)
272.         {
273.             idx2 = (idx2 + 1) % 256;
274.
275.             var_143 += layer2_rc4_state[idx2];
276.             var_144 = layer2_rc4_state[idx2];
277.             layer2_rc4_state[idx2] = layer2_rc4_state[var_143];
278.             layer2_rc4_state[var_143] = var_144;
279.
280.             uint8_t keyStreamByte = layer2_rc4_state[ (layer2_rc4_state[idx2] + layer2_rc4_state[var_143]) & 0xFF] &
    0xFF;
281.
282.             output[i] = layer_ptr[i] ^ keyStreamByte;
283.         }
284.     }
285.     else if(layer_num == 2)
286.     {
287.         //"Woot !! Smells good :)"
288.         if(strncmp(&ram[0x10+offset] , "V29vdCAhISBTbWVsbHMgZ29vZCA6KQ==", 0x20))
289.         {
290.             printf("Error: bad key2\n");
291.             exit(1);
292.         }
293.         free(output);
294.         output = NULL;
295.     }
296.     else if(layer_num == 0)
297.     {
298.         uint32_t* output32 = (uint32_t*) output;
299.
300.         output32[0] = ((uint32_t*)layer_ptr)[0] ^ output_key;
301.
302.         for(i=1; i < layer_length/4; i++)
303.         {
304.             output[i] = ((uint32_t*)layer_ptr)[i] ^ output32[i-1];
305.         }
306.         if( output_key == key_int_swapped)
307.         {
308.             printf("Error: bad key2\n");
309.             exit(1);
310.         }
311.     }
312.
313.     return output;
314. }
315.
316. int main(int argc, char** argv)
317. {
318.     int i = 0;
319.     char* output_filename;
320.     int encrypt_flag = 0;
321.     int bytes_read = 0;
322.     unsigned char* buf2;
323.     unsigned char* buf;
324.     int file_size;
325.     int fd;
326.     unsigned char md5_tocompare[16];
327.     unsigned char computed_hash[16];
328.     char key1[17];
329.     char key2[17];
330.     RC4_KEY rc4_key;
331.
332.     printf("--> SSTICRYPT <--\n");
333.     if(argc != 4)
334.     {
335.         printf("usage: %s [-d|-e] <key> <secure container>\n" \
336.                 "\t-d: uncrypt\n" \
337.                 "\t-e: crypt\n", argv[0]);
338.         exit(-1);
339.     }
340.     if(strlen(argv[2]) != 32)
```

```
    {
        printf("Error: key should be a 128-bits hexadecimal string\n");
        exit(-1);
    }

    if(!strcmp(argv[1], "-e"))
    {
        encrypt_flag = 1;
    }

    fd = open(argv[3], O_RDONLY);
    if(fd < 0)
    {
        perror("open");
        exit(-1);
    }
    file_size = lseek(fd, 0, SEEK_END);
    lseek(fd, 0, SEEK_SET);
    if(!encrypt_flag)
    {
        read(fd, md5_tocompare, 16);
        file_size -= 16;
    }
    buf = calloc(file_size, 1);
    if(buf == NULL)
    {
        exit(-1);
    }
    do
    {
        bytes_read += read(fd, buf, file_size);
    }while(bytes_read != file_size);

    close(fd);

    if (!encrypt_flag)
    {
        MD5(buf, file_size, computed_hash);
        if(memcmp(md5_tocompare, computed_hash, 16))
        {
            printf("Warning: MD5 mismatch for container\n");
        }
    }

    buf2 = calloc(file_size, 1);

    strncpy(key2, &argv[2][16], 16);
    key2[16] = '\0';
    strncpy(key1, argv[2], 16);
    key1[16] = '\0';

    printf("Using keys %s / %s ...\n", key1, key2);

    if(!strcmp(key1, "5132397062694567") && !strcmp(key2, "513239706269453d"))
    {
        printf("Error: You're a duck\n");
        exit(1);
    }
    if(!encrypt_flag)
    {
        check_key(key1, 1);
        check_key(key2, 2);
    }

    RC4_set_key(&rc4_key, 32, (unsigned char*) argv[2]);

    if(!encrypt_flag)
    {
        for(i=1; i < file_size; i++)
        {
            buf[i-1] ^= buf[i];
        }
    }
    RC4(&rc4_key, file_size, buf, buf2);

    if(encrypt_flag)
    {
        for(i=file_size-1; i > 0; i--)
        {
            buf2[i-1] ^= buf2[i];
        }
    }

    output_filename = calloc(strlen(argv[3]) + 5, 1);

    sprintf(output_filename, encrypt_flag ? "%s.enc" : "%s.dec", argv[3]);

    fd = open(output_filename, O_CREAT | O_WRONLY);
```

```
429.
430.     if(fd < 0)
431.     {
432.         perror("open");
433.         exit(-1);
434.     }
435.     MD5(buf2, file_size, computed_hash);
436.
437.     if(encrypt_flag)
438.     {
439.         write(fd, computed_hash, 16);
440.     }
441.     write(fd, buf2, file_size);
442.     close(fd);
443.
444.     return 0;
445. }
```

CHECK.PY

```
1.  #!/usr/bin/python
2.
3.  from math import floor, log
4.  from binascii import b2a_hex, a2b_hex
5.  import sys
6.  import struct
7.  import random
8.  import pickle
9.
10. class Bits ():
11. #__module__ = __name__
12.     ival = 0L
13.     size = 1
14.     mask = 1L
15.
16.     def __init__(self, v, size=None):
17.         if isinstance(v, Bits):
18.             self.ival = v.ival
19.             self.size = v.size
20.             self.mask = v.mask
21.         elif isinstance(v, int) or isinstance(v, long):
22.             if v:
23.                 self.ival = abs(v * 1L)
24.                 self.size = int(floor(log(self.ival)/log(2) + 1))
25.                 self.mask = (1L << self.size) - 1L
26.         elif isinstance(v, list):
27.             self.size = len(v)
28.             self.mask = (1L << self.size) - 1L
29.             for i, x in enumerate(v):
30.                 self[i] = x
31.         elif isinstance(v, str):
32.             self.size = len(v) * 8
33.             self.mask = (1L << self.size) - 1L
34.             l = map(ord, v)
35.             i = 0
36.             for o in l:
37.                 self[i:i+8] = Bits(o, 8).bitlist()[::-1]
38.                 i += 8
39.         if size: self.size = size
40.
41.     def __len__(self):
42.         return self.size
43.
44.     def bit(self, i):
45.         if i in range(self.size):
46.             return (self.ival >> i) & 1L
47.         elif -i in range(self.size + 1):
48.             return (self.ival >> (self.size + i)) & 1L
49.         else:
50.             raise IndexError
51.
52.     def __setattr__(self, field, v):
53.         if field == 'size':
54.             self.__dict__['size'] = v
55.             self.__dict__['mask'] = (1L << v) - 1L
56.         else:
57.             self.__dict__[field] = v
58.
59.     def __repr__(self):
60.         c = self.__class__
61.         l = self.size
62.         s = self.ival
63.         return '<%s instance with ival=%x (len=%d)>' % (c, s, l)
64.
65.     def __str__(self):
```

```
        'binary string representation, bit0 first.'
        s = ''
        for i in self:
            s = s + str(i)
        return s

    def __hex__(self):
        'byte string representation, bit0 first.'
        s = '%x' % self[::-1].ival
        return a2b_hex(s.rjust(self.size/4, '0'))

    def __cmp__(self, a):
        if not isinstance(a, Bits): raise AttributeError
        if self.size != a.size: raise ValueError
        return cmp(self.ival, a.ival)

    def __eq__(self, a):
        if isinstance(a, Bits): a = a.ival
        return self.ival == a

    def __ne__(self, a):
        if isinstance(a, Bits): a = a.ival
        return self.ival != a

    def __iter__(self):
        'bit iterator.'
        for x in range(self.size):
            yield self.bit(x)

    def __getitem__(self, i):
        'getitem defines b[i], b[i:j] and b[list] and returns a Bits instance'
        if isinstance(i, int):
            return Bits(self.bit(i), 1)
        elif isinstance(i, slice):
            return Bits(self.bitlist()[i])
        else:
            s = []
            for x in i:
                s.append(self.bit(x))
            return Bits(s)

    def __setitem__(self, i, v):
        '''''setitem defines
              b[i]=v with v in (0,1),
              b[i:j]=v
          and b[list]=v where
    v is iterable with range equals to that required by i:j or list,
    or v generates a Bits instance of desired length.'''
        if isinstance(i, int):
            assert v in (0,1)
            if i in range(self.size):
                if self.bit(i) == 1: self.ival -= (1L << i)
                self.ival += (v & 1L) << i
            elif -i in range(self.size + 1):
                p = self.size + i
                if self.bit(p) == 1: self.ival -= (1L << p)
                self.ival += (v & 1L) << p
            else:
                raise IndexError
        else:
            if isinstance(i, slice):
                start, stop, step = i.indices(self.size)
                r = range(start, stop, step)
            else:
                r = i
            try:
                assert len(r) == len(v)
                for j, b in zip(r,v):
                    self[j] = b
            except (TypeError, AssertionError):
                for j, b in zip(r, Bits(v, len(r))):
                    self[j] = b

    def __lshift__(self, i):
        res = Bits(self)
        res.ival = (res.ival << i) & res.mask
        return res
    def __rshift__(self, i):
        res = Bits(self)
        res.ival = (res.ival >> i) & res.mask
        return res
    def __invert__(self):
        res = Bits(self)
        res.ival = res.ival ^ res.mask
        return res

    def __and__(self, rvalue):
        if not isinstance(rvalue, Bits):
```

```
            obj = Bits(rvalue)
        else:
            obj = rvalue
        if self.size > obj.size:
            res = Bits(self)
        else:
            res = Bits(obj)
        res.ival = self.ival & obj.ival
        return res
    def __or__(self, rvalue):
        if not isinstance(rvalue, Bits):
            obj = Bits(rvalue)
        else:
            obj = rvalue
        if self.size > obj.size:
            res = Bits(self)
        else:
            res = Bits(obj)
        res.ival = self.ival | obj.ival
        return res
    def __xor__(self, rvalue):
        if not isinstance(rvalue, Bits):
            obj = Bits(rvalue)
        else:
            obj = rvalue
        if self.size > obj.size:
            res = Bits(self)
        else:
            res = Bits(obj)
        res.ival = self.ival ^ obj.ival
        return res

    def __rand__(self, lvalue):
        return self & lvalue
    def __ror__(self, lvalue):
        return self | lvalue
    def __rxor__(self, lvalue):
        return self ^ lvalue

    def __floordiv__(self, rvalue):
        'operator // is used for concatenation.'
        if not isinstance(rvalue, Bits):
            obj = Bits(rvalue)
        else:
            obj = rvalue
        return Bits(self.bitlist() + obj.bitlist())

    def bitlist(self):
        'return a list of bits (bit0 first)'
        return map(int, str(self))

    def hw(self):
        'hamming weight of the object (count of 1s).'
        return self.bitlist().count(1)

    def hd(self, other):
        'hamming distance to another object of same length.'
        if not isinstance(other, Bits):
            obj = Bits(other)
        else:
            obj = other
        if self.size != obj.size: raise ValueError
        return (self ^ obj).hw()


class ECB(object):
    'Electronic Code Book mode of Operation'

    def pad(self, M):
        m = '%s' % M
        n, p = divmod(len(m), 8)
        if p > 0:
            m += chr(8-p) * (8-p)
            n += 1
        return m

    def enc(self, M):
        n, p = divmod(len(M), 8)
        if p > 0:
            M+= chr(8-p) * (8-p)
            n += 1
        C = []
        for b in range(n):
            C.append(hex(self._cipher(Bits(M[0:8]), 1)))
            M = M[8:]
        assert len(M) == 0
        return ''.join(C)

```

```
    def dec(self, C):
        n, p = divmod(len(C), 8)
        assert p == 0
        M = []
        for b in range(n):
            M.append(hex(self._cipher(Bits(C[0:8]), -1)))
            C = C[8:]
        assert len(C) == 0
        return ''.join(M)


    def _cipher(self, B, direction):
        return B

def enc(K, M):
    assert M.size == 64
    k = PC1(K)
    blk = IP(M)
    L = blk[0:32]
    R = blk[32:64]
    for r in range(16):
        fout = F(R,k,r)
        L = L ^ fout
        L, R = R, L
    L, R = R, L
    C = Bits(0, 64)
    C[0:32] = L
    C[32:64] = R
    return IPinv(C)

def dec(K, C):
    assert C.size == 64
    k = PC1(K)
    blk = IP(C)
    L = blk[0:32]
    R = blk[32:64]
    for r in range(16)[::-1]:
        fout = F(R,k,r)
        L = L ^ fout
        L, R = R, L
    L, R = R, L
    M = Bits(0, 64)
    M[0:32] = L
    M[32:64] = R
    return IPinv(M)

def subkey(k, r):
    C = k[0:28]
    D = k[28:56]
    shifts = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]
    s = sum(shifts[:r + 1])
    C = (C >> s) | (C << (28-s))
    D = (D >> s) | (D << (28-s))
    return PC2(C // D)

def F(R, k, r):
    RE = E(R)
    Z = Bits(0,32)
    fk = subkey(k, r)
    s = RE ^ fk
    ri, ro = (0,0)
    for n in range(8):
        nri, nro = ri+6, ro+4
        x = s[ri:nri]
        i = x[(5,0)].ival
        j = x[(4,3,2,1)].ival
        Z[ro:nro] = Bits(S(n, (i << 4) + j), 4)[::-1]
        ri, ro = nri, nro
    return P(Z)

def IP(M):
    assert M.size == 64
    table = [57, 49, 41, 33, 25, 17, 9,  1,
        59, 51, 43, 35, 27, 19, 11, 3,
        61, 53, 45, 37, 29, 21, 13, 5,
        63, 55, 47, 39, 31, 23, 15, 7,
        56, 48, 40, 32, 24, 16, 8,  0,
        58, 50, 42, 34, 26, 18, 10, 2,
        60, 52, 44, 36, 28, 20, 12, 4,
        62, 54, 46, 38, 30, 22, 14, 6]
    return M[table]

def IPinv(M):
    assert M.size == 64
    table = [39,  7, 47, 15, 55, 23, 63, 31,
        38,  6, 46, 14, 54, 22, 62, 30,
        37,  5, 45, 13, 53, 21, 61, 29,
        36,  4, 44, 12, 52, 20, 60, 28,
```

```
        35,  3, 43, 11, 51, 19, 59, 27,
        34,  2, 42, 10, 50, 18, 58, 26,
        33,  1, 41,  9, 49, 17, 57, 25,
        32,  0, 40,  8, 48, 16, 56, 24]
    return M[table]

def PC1(K):
    table = [56, 48, 40, 32, 24, 16,  8,
          0, 57, 49, 41, 33, 25, 17,
          9,  1, 58, 50, 42, 34, 26,
         18, 10,  2, 59, 51, 43, 35,
         62, 54, 46, 38, 30, 22, 14,
          6, 61, 53, 45, 37, 29, 21,
         13,  5, 60, 52, 44, 36, 28,
         20, 12,  4, 27, 19, 11,  3 ]
    return K[table]

def PC2(K):
    assert K.size == 56
    table =  [ 13, 16, 10, 23,  0,  4,
         2, 27, 14,  5, 20,  9,
        22, 18, 11,  3, 25,  7,
        15,  6, 26, 19, 12,  1,
        40, 51, 30, 36, 46, 54,
        29, 39, 50, 44, 32, 47,
        43, 48, 38, 55, 33, 52,
        45, 41, 49, 35, 28, 31]
    return K[table]

def E(L):
    assert L.size == 32
    table = [31, 0, 1, 2, 3, 4,
             3,4,5,6,7,8,
             7,8,9,10,11,12,
             11,12,13,14,15,16,
             15,16,17,18,19,20,
             19,20,21,22,23,24,
             23,24,25,26,27,28,
             27,28,29,30,31,0]
    return L[table]

def P(s):
    assert s.size == 32
    table = [15, 6, 19, 20, 28, 11,
        27, 16, 0, 14, 22, 25,
        4, 17, 30, 9, 1, 7,
        23,13, 31, 26, 2, 8,
        18, 12, 29, 5, 21, 10,
        3, 24]
    return s[table]

def S(n, x):
    assert 0 <= n < 8
    assert 0 <= x < 64

    boxes = [
        [14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
         0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
         4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
         15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13],

        [15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
         3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
         0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
         13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9],

        [10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
         13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
         13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
         1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12],

        [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
         13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
         10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
         3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14],

        [2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
         14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
         4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
         11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3],

        [12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
         10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
         9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
         4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13],

        [4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
         13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
```

```
         1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
         6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12],

        [13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
         1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
         7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
         2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11],
    ]
    return Bits(boxes[n][x], 4)


class WhiteDES(ECB):

    def __init__(self, KT, tM1, tM2, tM3):
        self.KT = KT
        self.tM1 = tM1
        self.tM2 = tM2
        self.tM3 = tM3

    def FX(self, v):
        res = Bits(0, 96)
        for b in range(96):
            res[b] = (v & self.tM2[b]).hw() % 2
        return res

    def _cipher(self, M, d):
        assert M.size == 64
        if d == 1:
            blk = M[self.tM1]
            for r in range(16):
                t = 0
                for n in range(12):
                    nt = t + 8
                    blk[t:nt] = self.KT[r][n][blk[t:nt].ival]
                    t = nt
                blk = self.FX(blk)
            return blk[self.tM3]
        if d == -1:
            raise NotImplementedError


if __name__ == "__main__":
    WT = pickle.loads("""<instance d'objet WhiteDES sous forme de pickle>""")
    if len(sys.argv) == 1:
        print "Usage: python check.pyc <key>"
        print "   - key: a 64 bits hexlify-ed string"
        print "Example: python check.pyc 0123456789abcdef"
    else:
        K = Bits(a2b_hex(sys.argv[1]), 64)
        assert K[range(7,64,8)] == 175
        M = Bits(random.getrandbits(64), 64)



        if hex(WT._cipher(M, 1)) == hex(enc(K, M)):
            exit(0)
        else:
            exit(1)
```

WHITEBOX.PY

```
from check import *
import random

"""
http://eprint.iacr.org/2004/025.pdf
4 Attack on Split T-Box Output
"""

Etable = [31, 0, 1, 2, 3, 4,
             3,4,5,6,7,8,
             7,8,9,10,11,12,
             11,12,13,14,15,16,
             15,16,17,18,19,20,
             19,20,21,22,23,24,
             23,24,25,26,27,28,
             27,28,29,30,31,0]

"""
As part of the attack implementation, a reference
DES is created that produces first-round s-box re-
sults. All 64 6-bit s-box inputs (corresponding to
```

```
6 bits of message encrypted with a zero key) are
passed through all eight s-boxes and mapped back-
wards through EP and P1. The DES inputs cor-
responding to each of the 16 4-bit results for each
s-box are recorded as preimage sets. Each s-box has
16 zero-key preimage sets with 4 elements.
"""
def gen_preimages(sboxn):
    preimage_sets = {}
    L = Bits(random.getrandbits(32), 32)
    for i in xrange(64):
        b = Bits(i, 6)
        xorout = b[(5,3,2,1,0,4)]
        R = Bits(random.getrandbits(32), 32)
        for j in xrange(6):
            R[Etable[sboxn*6+j]] = xorout.bit(j)

        assert R[Etable][sboxn*6:(sboxn+1)*6] == xorout
        blk = L // R
        pre = IPinv(blk)
        sboxout = S(sboxn, i)
        preimage_sets.setdefault(sboxout.ival, []).append(pre)
    return preimage_sets

"""
return 64 bit DES key from round r subkey
8 bit guess is used to fill the missing bits when reverting from 48 bits => 56 bits
8 parity bits are set to 0
"""
def inv_subkey(sk, r, guess):
    pc1table = [56, 48, 40, 32, 24, 16,  8,
          0, 57, 49, 41, 33, 25, 17,
          9,  1, 58, 50, 42, 34, 26,
         18, 10,  2, 59, 51, 43, 35,
         62, 54, 46, 38, 30, 22, 14,
          6, 61, 53, 45, 37, 29, 21,
         13,  5, 60, 52, 44, 36, 28,
         20, 12,  4, 27, 19, 11,  3 ]
    pc2table =  [ 13, 16, 10, 23,  0,  4,
         2, 27, 14,  5, 20,  9,
        22, 18, 11,  3, 25,  7,
        15,  6, 26, 19, 12,  1,
        40, 51, 30, 36, 46, 54,
        29, 39, 50, 44, 32, 47,
        43, 48, 38, 55, 33, 52,
        45, 41, 49, 35, 28, 31]
    shifts = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]
    s = sum(shifts[:r + 1])
    bitsToGuess = [i for i in xrange(56) if not i in pc2table]
    invpc2 = Bits(0,56)
    for i in xrange(len(pc2table)):
        invpc2[pc2table[i]] = sk.bit(i)
    #fill missing bits
    invpc2[bitsToGuess] = guess
    C = invpc2[:28]
    D = invpc2[28:]

    C = (C << s) | (C >> (28-s))
    D = (D << s) | (D >> (28-s))
    k = C // D

    K = Bits(0, 64)
    for i in xrange(len(pc1table)):
        K[pc1table[i]] = k.bit(i)
    return K

#add attack implementation to WhiteDES class
class WhiteDES1(WhiteDES):
    #return T boxs output after 1 round
    def doOneRound(self, M):
        assert M.size == 64
        blk = M[self.tM1]
        for r in range(1):
            t = 0
            for n in range(12):
                nt = t + 8
                blk[t:nt] = self.KT[r][n][blk[t:nt].ival]
                t = nt
            return blk

    def getSBoxsInputs(self, M):
        R = IP(M)[32:64]
        sboxin = E(R)
        res = set()
        for sboxnum in xrange(8):
            #The T-box that is changed by both b1 and b4 for an s-box is the matching T-box,
            #only return sbox number if those bits are set in input
            for bitnum in [1,4]:
```

```
                if sboxin.bit(sboxnum*6 + bitnum):
                    res.add(sboxnum)
        return res

    def changedTBoxes(self, ref, other):
        res = set()
        for i in xrange(ref.size):
            if ref.bit(i) != other.bit(i):
                res.add(i / 8)
        return res

    """
    The first step of the attack is to identify which of the
    twelve T-boxes correspond to the eight s-boxes. This
    is a matter of encrypting a zero block through the
    first round of T-boxes, and then encrypting individ-
    ual bits and observing which T-box outputs change.
    For the bits b0b1b2b3b4b5passed into an s-box, bits
    b0and b5are passed as bypass bits in the T-box to
    create a bijection, b1and b4are produced as bypass
    bits for other T-boxes, and b2and b3are replicated
    in an arbitrary T-box as part of the replicated bits
    (those bits not duplicated by EP) for Rr. The T-box
    that is changed by both b1and b4for an s-box is the
    matching T-box, and in particular the 4-bit T-box
    output block that changes corresponds to the 4-bit
    s-box output. Our attack uses this process to build
    s-box output bitmasks.
    """
    def mapSboxes(self):
        zeroOut = self.doOneRound(Bits(0,64))
        xx={}
        for i in xrange(64):
            x = Bits(1 << i, 64)
            sb = self.getSBoxsInputs(x)
            if len(sb):
                print sb
                tout = self.doOneRound(x)
                tbox = self.changedTBoxes(zeroOut, tout)
                print tbox
                sb = list(sb)[0]
                if not xx.has_key(sb):
                    xx[sb] = tbox
                else:
                    xx[sb] = xx[sb].intersection(tbox)
        print xx

    def gen_key_pre(self, guess, sboxn):
        R = Bits(random.getrandbits(32), 32)
        L = Bits(random.getrandbits(32), 32)
        #reverse Expansion
        for j in xrange(6):
            R[Etable[sboxn*6+j]] = guess.bit(j)

        assert R[Etable][sboxn*6:(sboxn+1)*6] == guess
        blk = L // R
        return IPinv(blk)

    """
    For each candidate key, the 6 bits are reversed
    through EP and the DES initial permutation to get
    a plaintext that effectively cancels those key bits for
    that s-box. This key preimage is xored with the el-
    ements of a zero-key preimage set, and these values
    are passed into the white-box DES.
    The s-box out-put bitmask is used to isolate the results, and the
    four results are compared. If they are not all equal,
    then the candidate key bits are incorrect, and the al-
    gorithm moves on. If all four results are equal for all
    preimage sets, the candidate key bits are correct
    """
    def findSubkeyBitsForSbox(self, sboxn):
        preimages = gen_preimages(sboxn)
        ok=[]
        for guess in [Bits(i, 6) for i in xrange(64)]:
            keypreimage = self.gen_key_pre(guess, sboxn)
            z = []
            for sboxout, pres in preimages.items():
                s = set()
                for pre in pres:
                    res = self.doOneRound(pre ^ keypreimage)
                    #Tboxs are ordered for this instance (tbox0 = sbox0)
                    s.add(res[8*sboxn:8*(sboxn+1)][:4].ival)
                if len(s) != 1:
                    break
                z.append(s)
            if len(z) == 16:
                print "sub-subkey for sbox %d : %s" % (sboxn, guess)
```

```
                ok.append(guess)
        return ok

    def go(self):
        subkey1 = Bits(0, 48)
        for sb in xrange(8):
            g = self.findSubkeyBitsForSbox(sb)
            #2 possible results for sbox 4
            assert len(g) == 1 or (len(g)==2 and sb == 3)
            if sb==3: guesses_for_sbox4 = g
            subkey1[6*sb:6*(sb+1)] = g[0]
        subkey2 = Bits(subkey1)
        subkey2[6*3:6*(3+1)] = guesses_for_sbox4[1] #use alternate guess for sbox4
        print "Found subkeys :"
        print subkey1
        print subkey2
        print "Bruteforcing last 8 bits for 2 possible subkeys"
        M = Bits(random.getrandbits(64), 64)
        cipher = self._cipher(M, 1)
        for subkey in [subkey1, subkey2]:
            for guess in [Bits(i,8) for i in xrange(256)]:
                key = inv_subkey(subkey, 0, guess)
                if hex(enc(key, M)) == hex(cipher):
                    key[range(7,64,8)] = 175 #set parity bits expected by check.py
                    assert hex(enc(key, M)) == hex(cipher)
                    print "Key found \o/", key
                    print "Key =", hex(key).encode("hex")
                    return

"""
Found subkey 101101011000111101011010111001011101101000100111
Bruteforcing last 8 bits
111111000100000010000100111111100110011010101000010010101111100
Key found \o/ 111111010100000110000101111111111011001101010100101001010111111101
Key = fd4185ff66a94afd
"""
if __name__ == "__main__":
    ssticrypt = open("ssticrypt","rb").read()
    wbpickle = ssticrypt[0x7333:0x7333 + 0x3fee7]
    WT = pickle.loads(wbpickle)
    wb = WhiteDES1(WT.KT, WT.tM1, WT.tM2, WT.tM3)
    #wb.mapSboxes()    #TBox are not shuffled
    wb.go()
```

DECRYPT_LAYERS.PY

```
import struct

def decrypt_layer2(data, key):
    tab=[]
    for i in xrange(0,len(data), 4):
        tab.append(struct.unpack(">L",data[i:i+4])[0])
    tab[0] ^= key

    for i in xrange(1, len(tab)):
        tab[i] ^= tab[i-1]

    return "".join(map(lambda x:struct.pack(">L",x), tab))

#http://stackoverflow.com/questions/9773271/rc4-decryption-with-key-in-python
def decrypt_layer3(data, box):
    x,y = 0, 0
    out = []
    for char in data:
        x = (x + 1) % 256
        y = (y + box[x]) % 256
        box[x], box[y] = box[y], box[x]
        out.append(chr(ord(char) ^ box[(box[x] + box[y]) % 256]))
    return ''.join(out)

if __name__ == "__main__":
    ssticrypt = open("ssticrypt","rb").read()

    layer1 = ssticrypt[0x4888c:0x4888c + 0x65a]
    open("layer1.bin","wb").write(layer1)

    layer2 = ssticrypt[0x48ee8:0x48ee8 + 0xee2 + 2]

    layer2  = decrypt_layer2(layer2, 0x8cfad5fb)

    open("layer2_decrypted.bin","wb").write(layer2)

    layer3 = ssticrypt[0x481a8:0x481a8 + 0x6e3]

    layer3box = [   0xE3, 0xEF, 0x1D, 0x26, 0xC2, 0x00, 0x44, 0x2E, 0xB3, 0xF0, 0x10, 0x7E,
                    0xFC, 0x6B, 0x06, 0xD0, 0x31, 0xE4, 0x3E, 0x6D, 0x8F, 0x9D, 0xB0, 0x2A,
                    0xF6, 0x43, 0x7D, 0x53, 0x58, 0x15, 0x11, 0x19, 0x02, 0xDB, 0xA0, 0x8E,
                    0x78, 0x25, 0x04, 0x5B, 0xA4, 0xA2, 0x13, 0xD4, 0x0A, 0xBC, 0x6E, 0xD3,
                    0x8D, 0x85, 0xCD, 0x84, 0x81, 0x87, 0xA3, 0x3C, 0x57, 0xC0, 0xDC, 0xA6,
                    0x73, 0xEE, 0x17, 0x0D, 0x61, 0xAE, 0x36, 0x83, 0x8B, 0x7C, 0xA8, 0x51,
                    0xF2, 0xCE, 0x59, 0xFD, 0x91, 0x6A, 0x1A, 0x4F, 0x5A, 0x4E, 0x70, 0xF1,
                    0xAC, 0x98, 0x2F, 0xD6, 0x7B, 0x49, 0x76, 0xB9, 0x0E, 0xB4, 0xF8, 0x8A,
                    0x5C, 0x7A, 0xB8, 0x9E, 0xDA, 0xD8, 0x23, 0xED, 0xDD, 0x6C, 0x0B, 0x29,
                    0x3F, 0x64, 0x62, 0xFF, 0xE7, 0xD2, 0x2B, 0x77, 0xD7, 0x94, 0x48, 0x39,
                    0x66, 0xA9, 0x4D, 0x35, 0x8C, 0xB5, 0x28, 0x1C, 0x60, 0x9F, 0x37, 0x52,
                    0x63, 0xBD, 0x99, 0x75, 0x1E, 0x34, 0xFA, 0x5F, 0x09, 0x67, 0x16, 0x3A,
                    0xD9, 0x71, 0xF4, 0x80, 0xC6, 0xE0, 0x89, 0x79, 0x9A, 0xEB, 0x9B, 0x90,
                    0x46, 0xFE, 0xC5, 0xBF, 0x21, 0x6F, 0xE2, 0xF5, 0x56, 0xD1, 0xCF, 0x74,
                    0xB1, 0xA5, 0xE1, 0x47, 0xD5, 0xE6, 0x27, 0x86, 0x3D, 0x1F, 0xCA, 0xC1,
                    0x1B, 0x05, 0xB6, 0xCC, 0xAA, 0x4B, 0x69, 0xE9, 0x03, 0xC8, 0xEA, 0x12,
                    0xC4, 0xC7, 0x96, 0xCB, 0xC9, 0xE8, 0x55, 0x82, 0x95, 0xF7, 0x07, 0x20,
                    0x08, 0x0C, 0xBB, 0x5E, 0x93, 0xAB, 0x14, 0x3B, 0xBA, 0x97, 0x4A, 0x33,
                    0x7F, 0x54, 0x22, 0x9C, 0xDF, 0xDE, 0x42, 0x40, 0x0F, 0x24, 0xB7, 0x50,
                    0x30, 0x2C, 0xAD, 0x01, 0xFB, 0xF9, 0x32, 0x5D, 0x38, 0xAF, 0xC3, 0x18,
                    0x72, 0xE5, 0xF3, 0x45, 0x88, 0x2D, 0x65, 0xA7, 0x4C, 0xEC, 0x68, 0x41,
                    0xB2, 0xA1, 0x92, 0xBE
                    ]
    layer3 = decrypt_layer3(layer3, layer3box)

    open("layer3_decrypted.bin","wb").write(layer3)
```

SSTIC_VM.PY

```
import sys

def cast_word(s):
    return "*((uint16_t*)&%s)" % s

class Operand(object):
    def __init__(self, p):
        self.sz = p.get_n_bits(1)
        self.type = p.get_n_bits(2)

        if self.type == 0:
            self.reg = p.get_n_bits(4)
        elif self.type == 1: #immediate
            if self.sz == 0:
                self.imm = p.get_n_bits(8)
            else:
                self.imm = p.get_n_bits(16)
        elif self.type == 2: #memory
            self.memtype = p.get_n_bits(2)
            if self.memtype == 1: #immaddr
                self.memimm = p.get_n_bits(16)
            elif self.memtype == 2: #[reg]
                self.memoff = p.get_n_bits(16)
                self.memreg = p.get_n_bits(4)
            elif self.memtype == 0:
                self.memreg = p.get_n_bits(4)
            elif self.memtype == 3:
                raise Exception("memtype = 3")
        else:
            self.imm =  p.get_n_bits(16)
            self.reg = p.get_n_bits(4)
            self.unk = p.get_n_bits(6)
            self.opname = "xorrol"

    def toC(self, idx):
        if self.type == 2:
            if self.memtype == 0:
                if self.sz == 0:
                    return "ram[r%d]" % self.memreg
                return cast_word("ram[r%d]" % self.memreg)
            elif self.memtype == 1:
                if self.sz == 0:
                    return "ram[0x%x]" % self.memimm
                return cast_word("ram[0x%x]" % self.memimm)
            elif self.memtype == 2:
                if self.sz == 0:
                    return "ram[0x%x + r%d]" % (self.memoff, self.memreg)
                return cast_word("ram[0x%x + r%d]" % (self.memoff, self.memreg))
        elif self.type == 3:
            return "operand3_read(0x%x, r%d, 0x%x)" % (self.imm, self.reg, self.unk)
        return str(self)

    def __str__(self):
        if self.type == 0:
            return "r%d" % self.reg
        elif self.type == 1:
            return "0x%x" % self.imm
        elif self.type == 2:
            if self.memtype == 0:
                if self.sz == 0:
                    return "b[r%d]" % self.memreg
                return "[r%d]" % self.memreg
            elif self.memtype == 1:
                if self.sz == 0:
                    return "b[0x%x]" % self.memimm
                return "[0x%x]" % self.memimm
            elif self.memtype == 2:
                return "%d[0x%x + r%d]" % (self.sz, self.memoff, self.memreg)
            else:
                return "memtype %d" % self.memtype
        else:
            return "[op3 0x%x r%d %d %x]" % (self.imm, self.reg, self.unk, self.sz)

class Instruction(object):
    def __init__(self, p):
        self.opname = "unk"
        self.operands = []
        self.eip = p.get_pc()
        self.b1 = b1 = p.get_n_bits(1)
        cond = p.get_n_bits(8)
        if not b1:
            cond >>= 4
        ccs = ["Z","NZ","C","NC","S","NS","O","NO","A","BE","G","GE","L","LE","",""]
        self.ccode =  ccs[cond & 0xf]
```

```
85.         if self.ccode != "":
86.             self.ccode += str(self.b1)
87.         op = p.get_n_bits(3)
88.         self.set_flags = p.get_n_bits(1)
89.
90.         if op == 0:
91.             self.opname = "and"
92.             self.operands.append(Operand(p))
93.             self.operands.append(Operand(p))
94.         elif op == 1:
95.             self.opname = "or"
96.             self.operands.append(Operand(p))
97.             self.operands.append(Operand(p))
98.         elif op == 2:
99.             self.opname = "not"
100.            self.operands.append(Operand(p))
101.        elif op == 3:
102.            direction = p.get_n_bits(1)
103.            self.opname = ["shl", "shr"][direction]
104.            n = p.get_n_bits(8)
105.            self.operands.append(Operand(p))
106.            self.operands.append(n)
107.        elif op == 4:
108.            self.opname = "mov"
109.            self.operands.append(Operand(p))
110.            self.operands.append(Operand(p))
111.        elif op == 5:
112.            #raise Exception("XXX opcode 5")
113.            self.opname = "op5"
114.            self.operands.append(Operand(p))
115.        elif op == 7 or op == 6:
116.            self.opname = "j"
117.            dst_bytes, dst_bits = 0, 0
118.            if op == 6:
119.                self.opname = "call"
120.            r12 = p.get_n_bits(6)
121.            r10 = p.get_n_bits(3)
122.            if r12 == 0:
123.                op = Operand(p)
124.                #print 'abs', op
125.                assert op.type == 1
126.                dst_bytes = op.imm
127.                dst_bits = r10
128.                jump_direction = "abs"
129.            else:
130.                jump_direction = r12 & 0x20
131.                bytes_to_add = r12 & 0x1f
132.                if jump_direction == 0:
133.                    r4 = p.pc_bits + r10
134.                    dst_bytes = p.pc_bytes + bytes_to_add + r4/8
135.                    dst_bits = r4 & 0x7
136.                    jump_direction = "forward"
137.                else:
138.                    dst_bytes = p.pc_bytes - bytes_to_add
139.                    dst_bits = p.pc_bits
140.                    if p.pc_bits >= r10:
141.                        dst_bytes -= 1
142.                        dst_bits += 8
143.                    dst_bits -= r10
144.                    jump_direction = "back"
145.            self.operands.append("%x:%d %s" % (dst_bytes,dst_bits, jump_direction))
146.
147.    def __str__(self):
148.        ops = ", ".join(map(str, self.operands))
149.        comments = ""
150.        opname = self.opname + self.ccode
151.        if self.set_flags:
152.            opname += "f" + str(self.b1)
153.        opname = opname.ljust(6, " ")
154.        return "%s %s %s       %s" % (self.eip, opname, ops, comments)
155.
156.    def toC(self):
157.        operands = []
158.        for o in self.operands:
159.            if isinstance(o, Operand):
160.                o = o.toC(len(operands))
161.            operands.append(str(o))
162.
163.        if isinstance(self.operands[0], Operand) and self.operands[0].type == 3:
164.            if self.opname in { "or": " | ", "and": " & ", "shr": ">>", "shl": "<<"}.keys():
165.                dd = operands[0] + { "or": " | ", "and": " & ", "shr": ">>", "shl": "<<"}[self.opname] + operands[1]

166.            elif self.opname  == "not":
167.                dd = "~" + operands[0]
168.            else:
169.                dd = operands[1]
170.            z = "operand3_write(0x%x, r%d, 0x%x, %s);" % (self.operands[0].imm, self.operands[0].reg, self.operands[0
    ].unk, dd)
```

```
        elif self.opname in ["mov", "or","and","shr","shl"]:
            z = operands[0] + {"mov": " = ", "or": " |= ", "and": " &= ", "shr": " >>= ", "shl": " <<= "}[self.opname
    ] +  operands[1] + ";"
        elif self.opname == "not":
            z = "%s = ~%s;" % (operands[0], operands[0])
        else:
            z = "//FIXME " + str(self)
        return z.ljust(80, " ") + "//"+ str(self)

class SSTICVM(object):
    def __init__(self, filename):
        self.d = open(filename,"rb").read()
        self.pc_bytes = 0
        self.pc_bits = 0

    def get_n_bits(self, n):
        if n <= (8-self.pc_bits):
            b = ord(self.d[self.pc_bytes])
            res = (b >> (8-self.pc_bits-n))
        else:
            b = ord(self.d[self.pc_bytes])
            res = ((b << (self.pc_bits)) & 0xff) >> self.pc_bits
            z = n - (8-self.pc_bits)
            i=1
            while z > 7:
                b = ord(self.d[self.pc_bytes+i])
                res = (res << 8) | b
                z -= 8
                i += 1
            b = ord(self.d[self.pc_bytes+i])
            res = (res << z) | (b >> (8-z))

        r4 = self.pc_bits + n
        self.pc_bytes += (r4/8)
        self.pc_bits = r4 & 0x7
        return res & ((1L << n)-1)

    def get_pc(self):
        return "%04x:%d     " % (self.pc_bytes, self.pc_bits)

    def disassemble(self, decomp):
        while ((self.pc_bytes*8 + self.pc_bits) /8  < len(self.d)):
            try:
                ins = Instruction(self)
            except IndexError:
                break
            if decomp:  print ins.toC()
            else:       print ins

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print "Usage: %s [-decomp] layer_file" % sys.argv[0]
        exit(0)
    if len(sys.argv) == 2:
        p = SSTICVM(sys.argv[1])
        p.disassemble(False)
    else:
        p = SSTICVM(sys.argv[2])
        p.disassemble(True)

```

LAYER1.C

```
1.  //gcc -Wall -O3 layer1.c -o layer1
2.  #include <stdio.h>
3.  #include <stdint.h>
4.
5.  uint16_t r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13,r14;
6.  uint16_t var_e3b4, var_c7f2, var_c882, var_f400, var_f31c,
7.      var_e99c, var_fc70, var_cac6, var_fe0c, var_e2ca,
8.      var_d68e, var_d760, var_1cb4, var_da1e, var_f26c, var_dd18,
9.      var_f9f8, var_fedc, var_c75e, var_d89e, var_d124, var_cf9a,
10.     var_e5e2, var_d4ac, var_cb26, var_deb4, var_f9e4,
11.     var_c1b4, var_dd8c, var_cbd4, var_f4ba, var_ea36,
12.     var_e614, var_f82e, var_fbce, var_f2ce, var_dd4e,
13.     var_c6c6, var_d1e4, var_e4ca, var_fa7a, var_fbc0,
14.     var_c33c, var_fb6a, var_f16c, var_c2ce, var_e6ea,
15.     var_c9c8, var_c82a, var_c71e, var_fd20, var_c1a4,
16.     var_d568, var_d800, var_e6b8, var_f0da, var_dfd0,
17.     var_e738, var_d25c, var_cb1e, var_8000, var_8002;
18.
19. uint16_t carry;
20.
21. uint16_t check_key1(uint16_t var_a000, uint16_t var_a002)
22. {
23.     r12 = var_a000;                         //0000:0
24.     r3 = var_a002;                          //0005:1
25.     r0 = r12;                               //000a:2
26.     r0 >>= 8;                               //000d:5
27.     r13 = r0;                               //0011:2
28.     r13 &= r12;                             //0014:5
29.     var_e3b4 = r0;                          //0018:0
30.     var_e3b4 |= r12;                        //001d:1
31.     var_e3b4 = ~var_e3b4;                   //0022:2
32.     r13 |= var_e3b4;                        //0026:4
33.     r13 = ~r13;                             //002b:5
34.     r6 = r13;                               //002e:1
35.     r5 = r6;                                //0031:4
36.     var_c7f2 = 0x9577;                      //0034:7
37.     var_c7f2 = ~var_c7f2;                   //003b:4
38.     r5 |= var_c7f2;                         //003f:6
39.     r5 = ~r5;                               //0044:7
40.     var_c882 = r6;                          //0047:3
41.     var_f400 = 0x9577;                      //004c:4
42.     var_f400 = ~var_f400;                   //0053:1
43.     var_c882 &= var_f400;                   //0057:3
44.     r5 |= var_c882;                         //005e:2
45.     r6 = r5;                                //0063:3
46.     r1 = r3;                                //0066:6
47.     r1 >>= 8;                               //006a:1
48.     var_f31c = r1;                          //006d:6
49.     var_e99c = r3;                          //0072:7
50.     var_e99c = ~var_e99c;                   //0078:0
51.     var_f31c &= var_e99c;                   //007c:2
52.     r5 = r1;                                //0083:1
53.     var_fc70 = r3;                          //0086:4
54.     var_fc70 = ~var_fc70;                   //008b:5
55.     r5 |= var_fc70;                         //008f:7
56.     var_fc70 = r5;                          //0095:0
57.     var_fc70 = ~var_fc70;                   //009a:1
58.     var_f31c |= var_fc70;                   //009e:3
59.     r8 = var_f31c;                          //00a5:2
60.     var_cac6 = r12;                         //00aa:3
61.     var_cac6 &= 0xff;                       //00af:4
62.     r1 = var_cac6;                          //00b6:1
63.     var_fe0c = r8;                          //00bb:2
64.     var_fe0c &= r1;                         //00c0:3
65.     r5 = r8;                                //00c5:4
66.     r5 |= r1;                               //00c8:7
67.     r5 = ~r5;                               //00cc:2
68.     var_fe0c |= r5;                         //00ce:6
69.     var_fe0c = ~var_fe0c;                   //00d3:7
70.     r8 = var_fe0c;                          //00d8:1
71.     var_e2ca = r1;                          //00dd:2
72.     var_e2ca <<= 8;                         //00e2:3
73.     r1 = var_e2ca;                          //00e7:6
74.     var_d68e = r8;                          //00ec:7
75.     var_d68e = ~var_d68e;                   //00f2:0
76.     var_d68e |= r1;                         //00f6:2
77.     var_d68e = ~var_d68e;                   //00fb:3
78.     var_d760 = r8;                          //00ff:5
79.     var_d760 = ~var_d760;                   //0104:6
80.     var_d760 &= r1;                         //0109:0
81.     var_d68e |= var_d760;                   //010e:1
82.     r8 = var_d68e;                          //0115:0
83.     var_c1b4 = r0;                          //011a:1
84.     var_c1b4 <<= 8;                         //011f:2
```

```
    r1 = var_c1b4;                          //0124:5
    var_da1e = r8;                          //0129:6
    var_da1e |= r1;                         //012e:7
    var_da1e = ~var_da1e;                   //0134:0
    var_f26c = r8;                          //0138:2
    var_f26c &= r1;                         //013d:3
    var_da1e |= var_f26c;                   //0142:4
    var_da1e = ~var_da1e;                   //0149:3
    r8 = var_da1e;                          //014d:5
    r5 = r8;                                //0152:6
    r5 &= r0;                               //0156:1
    r5 = ~r5;                               //0159:4
    var_dd18 = r8;                          //015c:0
    var_dd18 |= r0;                         //0161:1
    r5 &= var_dd18;                         //0166:2
    r8 = r5;                                //016b:3
    var_f9f8 = r8;                          //016e:6
    var_fedc = 0xae4d;                      //0173:7
    var_fedc = ~var_fedc;                   //017a:4
    r13 = 0x1175;                           //017e:6
    var_c75e = 0x1;                         //0183:5
    var_d89e = var_fedc;                    //018a:2

    do{
        r5 = var_c75e;                          //0191:1
        r5 = ~r5;                               //0196:2
        r5 |= var_d89e;                         //0198:6
        var_d124 = var_c75e;                    //019d:7
        var_cf9a = var_d89e;                    //01a4:6
        var_cf9a = ~var_cf9a;                   //01ab:5
        var_d124 |= var_cf9a;                   //01af:7
        r5 &= var_d124;                         //01b6:6
        r5 = ~r5;                               //01bb:7
        var_e5e2 = r5;                          //01be:3
        var_c75e &= var_d89e;                   //01c3:4
        var_d89e = var_c75e;                    //01ca:3
        var_c75e = var_e5e2;                    //01d1:2
        carry = var_d89e & 0x8000;
        var_d89e <<= 1;                         //01d8:1
        if(carry)
        {
        r13 |= 0x81a4;                          //01dd:4
        }
        var_d89e &= var_d89e;                   //01e2:3
    }while(var_d89e); //01e9:2      jNZ    191:1               //01e9:2

    r13 = ~r13;                             //01ee:3
    r13 <<= 1;                              //01f0:7
    var_fedc = var_c75e;                    //01f4:4
    r5 = 0x16b8;                            //01fb:3
    var_d4ac = var_f9f8;                    //0200:2
    var_cb26 = var_fedc;                    //0207:1
    do{
        var_deb4 = var_d4ac;                    //020e:0
        var_deb4 = ~var_deb4;                   //0214:7
        var_deb4 |= var_cb26;                   //0219:1
        var_deb4 = ~var_deb4;                   //0220:0
        var_f9e4 = var_d4ac;                    //0224:2
        var_f9e4 = ~var_f9e4;                   //022b:1
        var_f9e4 &= var_cb26;                   //022f:3
        var_deb4 |= var_f9e4;                   //0236:2
        r13 = var_deb4;                         //023d:1
        var_d4ac &= var_cb26;                   //0242:2
        var_cb26 = var_d4ac;                    //0249:1
        var_d4ac = r13;                         //0250:0
        carry = var_cb26 & 0x8000;
        var_cb26 <<= 1;                         //0255:1
        if(carry)
        {
        r5 |= 0xd8bf;                           //025a:4
        }var_cb26 &= var_cb26;                   //025f:3
    }while(var_cb26);//    0266:2      jNZ    20e:0               //0266:2

    r5 = ~r5;                               //026b:3
    r5 <<= 1;                               //026d:7
    var_f9f8 = var_d4ac;                    //0271:4
    var_f9f8 &= var_f9f8;                   //0278:3

    if(var_f9f8 == 0)               //027f:2      jNZ    560:0               //027f:2
    {
        var_fa7a = 0x539;                       //0284:3
        var_fa7a = ~var_fa7a;                   //028b:0
        var_dd8c = 0xc9b;                       //028f:2
        var_cbd4 = 0x1;                         //0295:7
        var_f4ba = var_fa7a;                    //029c:4
        do
        {    var_ea36 = var_cbd4;                   //02a3:3
            var_e614 = var_f4ba;                   //02aa:2
```

```
            var_e614 = ~var_e614;                   //02b1:1
            var_ea36 &= var_e614;                   //02b5:3
            r5 = var_cbd4;                          //02bc:2
            var_f82e = var_f4ba;                    //02c1:3
            var_f82e = ~var_f82e;                   //02c8:2
            r5 |= var_f82e;                         //02cc:4
            var_f82e = r5;                          //02d1:5
            var_f82e = ~var_f82e;                   //02d6:6
            var_ea36 |= var_f82e;                   //02db:0
            r13 = var_ea36;                         //02e1:7
            var_cbd4 &= var_f4ba;                   //02e7:0
            var_f4ba = var_cbd4;                    //02ed:7
            var_cbd4 = r13;                         //02f4:6
            carry = var_f4ba & 0x8000;
            var_f4ba <<= 1;                         //02f9:7
            if(carry)
            {var_dd8c |= 0xcd8e;                     //02ff:2
            }var_f4ba &= var_f4ba;                   //0305:7
        }while(var_f4ba);//    030c:6      jNZ    2a3:3               //030c:6
        var_dd8c = ~var_dd8c;                   //0311:7
        var_dd8c <<= 1;                         //0316:1
        var_fa7a = var_cbd4;                    //031b:4
        r13 = 0x3167;                           //0322:3
        var_fbc0 = r6;                          //0327:2
        var_f2ce = var_fa7a;                    //032c:3
        do{
            r5 = var_fbc0;                          //0333:2
            var_dd4e = var_f2ce;                    //0338:3
            var_dd4e = ~var_dd4e;                   //033f:2
            r5 &= var_dd4e;                         //0343:4
            var_c6c6 = var_fbc0;                    //0348:5
            var_c6c6 = ~var_c6c6;                   //034f:4
            var_c6c6 &= var_f2ce;                   //0353:6
            r5 |= var_c6c6;                         //035a:5
            var_d1e4 = r5;                          //035f:6
            var_fbc0 &= var_f2ce;                   //0364:7
            var_f2ce = var_fbc0;                    //036b:6
            var_fbc0 = var_d1e4;                    //0372:5
            carry = var_f2ce & 0x8000;
            var_f2ce <<= 1;                         //0379:4
            if(carry)
            {r13 |= 0xdcba;                          //037e:7
            }var_f2ce &= var_f2ce;                   //0383:6
        }while(var_f2ce);//    038a:5      jNZ    333:2               //038a:5
        r13 = ~r13;                             //038f:6
        r13 <<= 1;                              //0392:2
        r6 = var_fbc0;                          //0395:7
        var_f2ce = r6;                          //039b:0
        var_d1e4 = 0x6b14;                      //03a0:1
        var_d1e4 = ~var_d1e4;                   //03a6:6
        var_e4ca = 0x6d7b;                      //03ab:0
        var_c33c = 0x1;                         //03b1:5
        var_fb6a = var_d1e4;                    //03b8:2
        do
        {    r5 = var_c33c;                         //03bf:1
            r5 = ~r5;                               //03c4:2
            r5 |= var_fb6a;                         //03c6:6
            r5 = ~r5;                               //03cb:7
            var_f16c = var_c33c;                    //03ce:3
            var_f16c = ~var_f16c;                   //03d5:2
            var_f16c &= var_fb6a;                   //03d9:4
            r5 |= var_f16c;                         //03e0:3
            var_c2ce = r5;                          //03e5:4
            var_c33c &= var_fb6a;                   //03ea:5
            var_fb6a = var_c33c;                    //03f1:4
            var_c33c = var_c2ce;                    //03f8:3
            carry = var_fb6a & 0x8000;
            var_fb6a <<= 1;                         //03ff:2
            if(carry)
            {var_e4ca |= 0xa14f;                     //0404:5
            }var_fb6a &= var_fb6a;                   //040b:2
        }while(var_fb6a);//    0412:1      jNZ    3bf:1               //0412:1
        var_e4ca = ~var_e4ca;                   //0417:2
        var_e4ca <<= 1;                         //041b:4
        var_d1e4 = var_c33c;                    //0420:7
        var_e6ea = 0x4d27;                      //0427:6
        var_c9c8 = var_f2ce;                    //042e:3
        var_c82a = var_d1e4;                    //0435:2
        do
        {    var_c71e = var_c9c8;                   //043c:1
            var_c71e |= var_c82a;                   //0443:0
            r5 = var_c9c8;                          //0449:7
            r5 &= var_c82a;                         //044f:0
            r5 = ~r5;                               //0454:1
            var_c71e &= r5;                         //0456:5
            r13 = var_c71e;                         //045b:6
            var_c9c8 &= var_c82a;                   //0460:7
            var_c82a = var_c9c8;                    //0467:6
```

```
261.             var_c9c8 = r13;                          //046e:5
262.             carry = var_c82a & 0x8000;
263.             var_c82a <<= 1;                          //0473:6
264.             if(carry)
265.             {var_e6ea |= 0xd149;                      //0479:1
266.             }var_c82a &= var_c82a;                    //047f:6
267.         }while(var_c82a);//    0486:5      jNZ    43c:1                //0486:5
268.         var_e6ea = ~var_e6ea;                    //048b:6
269.         var_e6ea <<= 1;                          //0490:0
270.         var_f2ce = var_c9c8;                     //0495:3
271.         var_f2ce &= var_f2ce;                    //049c:2
272.         if(!var_f2ce)
273.         {
274.             var_fd20 = 0x84c;                        //04a3:1
275.             var_c1a4 = 0xdead;                       //04a9:6
276.             var_d568 = r8;                           //04b0:3
277.         }
278.
279.         do
280.         {
281.             if(!var_f2ce){
282.             var_d800 = var_c1a4;                     //04b5:4
283.             var_d800 = ~var_d800;                    //04bc:3
284.             var_d800 |= var_d568;                    //04c0:5
285.             var_d800 = ~var_d800;                    //04c7:4
286.             var_e6b8 = var_c1a4;                     //04cb:6
287.             var_e6b8 = ~var_e6b8;                    //04d2:5
288.             var_e6b8 &= var_d568;                    //04d6:7
289.             var_d800 |= var_e6b8;                    //04dd:6
290.             var_f0da = var_d800;                     //04e4:5
291.             var_c1a4 &= var_d568;                    //04eb:4
292.             var_d568 = var_c1a4;                     //04f2:3
293.             var_c1a4 = var_f0da;                     //04f9:2
294.             }
295.             carry = var_d568 & 0x8000;
296.             var_d568 <<= 1;                          //0500:1
297.             if(carry)
298.             {
299.                 var_fd20 |= 0xb4fd;                      //0505:4
300.             }
301.             var_d568 &= var_d568;                    //050c:1
302.             if(var_f2ce)
303.                 break;
304.         }while(var_d568);
305.             //0513:0      jNZ    51a:7                //0513:0
306.             //0515:6      jNZ    4b5:4                //0515:6
307.         //}while(var_d568);
308.         var_fd20 = ~var_fd20;                    //051a:7
309.         var_fd20 <<= 1;                          //051f:1
310.         if(!var_f2ce)
311.         {
312.             //var_a000 = var_c1a4;                     //0524:4
313.             var_dfd0 = r6;                           //052b:3
314.             var_e738 = 0xbeef;                       //0530:4
315.             var_e738 = ~var_e738;                    //0537:1
316.             var_dfd0 &= var_e738;                    //053b:3
317.             var_d25c = r6;                           //0542:2
318.             var_d25c = ~var_d25c;                    //0547:3
319.             var_d25c &= 0xbeef;                      //054b:5
320.             var_dfd0 |= var_d25c;                    //0552:2
321.             //var_a002 = var_dfd0;                     //0559:1
322.             printf("Output: 0x%x 0x%x\n", var_c1a4, var_dfd0);
323.             return 1;
324.         }
325.         //printf("%x %x %x %x\n", var_c1a4, var_a000, var_dfd0, var_a002);
326.         //return (var_c1a4 !=   var_a000) && (var_dfd0 != var_a002);
327.     }
328.     var_cb1e = r1;                           //0560:0
329.     var_cb1e |= r1;                          //0565:1
330.     var_8000 = 0xdead;                       //056a:2
331.     var_8002 = 0xbeef;                       //0570:7
332.     return 0;
333. }
334.
335. int main(int argc, char** argv)
336. {
337.     uint32_t i;
338.     for(i=0xffffffff; i > 0 ; i--)
339.     {
340.         if(!(i % 0x1000000))
341.         {
342.             printf("0x%x\n", i);
343.         }
344.         if(check_key1(i & 0xffff, (i >> 16) & 0xffff))
345.         {
346.             printf("Key? 0x%x 0x%x\n", i & 0xffff, (i >> 16) & 0xffff);
347.             break;
348.         }
```

```
    }
    return 0;
}
```

LAYER2.C

```
//gcc -Wall -O3 layer2.c -o layer2
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

unsigned char blob1[256] = {
    0x61, 0x48, 0xE9, 0xE8, 0x58, 0x27, 0xBC, 0x92, 0x6E, 0xD1, 0xA3, 0x7D,
    0x78, 0xED, 0x9C, 0x84, 0xCA, 0x1F, 0xCB, 0x65, 0x8A, 0x96, 0xA8, 0x5C,
    0x2C, 0x4E, 0xD5, 0x8C, 0x72, 0x8F, 0x80, 0x8F, 0xFE, 0x04, 0xAA, 0x84,
    0x28, 0x0F, 0x26, 0x2F, 0x9C, 0x4F, 0x8D, 0x17, 0x42, 0xE7, 0x19, 0xB7,
    0xBD, 0x38, 0xB4, 0x18, 0x7D, 0xCC, 0x25, 0x5D, 0xA7, 0x4E, 0x68, 0x12,
    0xEE, 0xB4, 0xDC, 0x64, 0x6F, 0x13, 0x79, 0x9F, 0x4D, 0x17, 0x89, 0x32,
    0xA0, 0x20, 0xBB, 0xCD, 0xF1, 0x10, 0xA1, 0x3A, 0x21, 0xD2, 0x4D, 0x5D,
    0xC8, 0xD0, 0x14, 0x84, 0x60, 0x96, 0x3A, 0x19, 0x24, 0xCD, 0x53, 0x8F,
    0xB5, 0xA5, 0x6B, 0x67, 0xD1, 0xBE, 0xC6, 0x66, 0xF1, 0x8C, 0x6D, 0xF7,
    0x76, 0xA2, 0x0C, 0xAD, 0x89, 0xEB, 0xDB, 0x9E, 0x49, 0x5A, 0xD0, 0x31,
    0xD7, 0x61, 0x71, 0xEC, 0xD3, 0x6B, 0xCC, 0x54, 0x9F, 0x9F, 0x83, 0xD7,
    0xDA, 0x5E, 0x7D, 0xAF, 0xF9, 0x0F, 0x07, 0xD5, 0x38, 0x75, 0x63, 0x10,
    0xE7, 0x5A, 0x5D, 0x61, 0x99, 0x36, 0x66, 0x26, 0x98, 0xE6, 0x9B, 0x7C,
    0xA9, 0xE8, 0x1D, 0x35, 0x82, 0x52, 0x39, 0x75, 0x0B, 0x70, 0x0E, 0x2A,
    0xB2, 0x95, 0xF3, 0xCA, 0xE8, 0xF7, 0x11, 0xEE, 0x0C, 0x2F, 0x23, 0xE1,
    0x48, 0x80, 0xB5, 0x9E, 0xD7, 0x88, 0x3D, 0xC7, 0x88, 0x3C, 0x97, 0x74,
    0x08, 0x16, 0x39, 0x75, 0xE6, 0xCB, 0x80, 0xE1, 0x65, 0xD6, 0x3D, 0xB6,
    0x80, 0xB8, 0x9E, 0x39, 0x24, 0x39, 0x79, 0x05, 0xEB, 0x8F, 0xFD, 0x53,
    0x59, 0x97, 0xCA, 0x8B, 0x79, 0x49, 0x53, 0x26, 0xAE, 0x96, 0x84, 0xEB,
    0x7A, 0x4D, 0xB4, 0x64, 0xFE, 0xCA, 0xA8, 0xD5, 0xFC, 0x39, 0x2E, 0x61,
    0x19, 0x16, 0xC2, 0x64, 0x15, 0x23, 0x1E, 0xD8, 0x96, 0xE9, 0x34, 0x47,
    0xA5, 0x8F, 0x66, 0xC4
};

uint8_t ram[0x10000]={0};

static unsigned int rol16(unsigned int i, int c) {
    return ((i << c) & 65535) | ((unsigned int)(i & 65535) >> (16 - c));
}
static unsigned int ror16(unsigned int i, int c) {
    return ((unsigned int)(i & 65535) >> c) | ((i << (16 - c)) & 65535);
}

uint16_t write3(uint16_t imm16, uint16_t reg, uint16_t imm6, uint16_t val)
{
    uint16_t r5 = ((((imm6 << 6) + imm6) << 4) + imm6) ^ 0x464d;
    uint16_t r2 = (imm16 ^ 0x6c38);
    uint16_t r6 = 0;
    uint16_t r7 = 0;
    imm16 += reg;
    reg += 2;
    do
    {
        r7 = r6;
        r5 = rol16(r5,1);
        r2 = ror16(r2,2) + r5;
        r5 += 2;
        r6 = r2 ^ r5;
        r6 = ((r6 >> 8 ) & 0xff) ^ (r6 & 0xff);
        reg--;
    }while(reg);

    r6 <<= 8;
    r6 |= r7;

    //printf("write at %x = %x\n", imm16, val ^ r6);
    *((uint16_t*)&ram[imm16]) = val ^ r6;
    return 0;
}

uint16_t read3(uint16_t imm16, uint16_t reg, uint16_t imm6)
{
    uint16_t r5 = ((((imm6 << 6) + imm6) << 4) + imm6) ^ 0x464d;
    uint16_t r2 = (imm16 ^ 0x6c38);
    uint16_t r6 = 0;
    uint16_t r7 = 0;
```

```
    imm16 += reg;
    reg += 2;
    do
    {
        r7 = r6;
        r5 = rol16(r5,1);
        r2 = ror16(r2,2) + r5;
        r5 += 2;
        r6 = r2 ^ r5;
        r6 = ((r6 >> 8 ) & 0xff) ^ (r6 & 0xff);
        reg--;
    }while(reg);

    r6 <<= 8;
    r6 |= r7;

    uint16_t res = *((uint16_t*)&ram[imm16]) ^ r6;
    //printf("read %x = %x\n", imm16, res);
    return res;
}
uint16_t r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13,r14;
uint16_t zz;

uint16_t check_key2(uint16_t var_a000, uint16_t var_a002)
{
    memcpy(&ram[0xA000], &var_a000, 2);
    memcpy(&ram[0xA002], &var_a002, 2);
    memcpy(&ram[0xA010], blob1, 256);

    //0000:0      mov    r4, 0x0
    r4 = 0x0;
    //0004:7      mov    [op3 0xca5c r4 15 1], r11
    write3(0xca5c, r4, 0xf, r11);
    //000b:0      or     [op3 0xca5c r4 15 1], r10
    write3(0xca5c, r4, 0xf, read3(0xca5c, r4, 0xf) | r10);
    //0011:1      mov    [op3 0xca5c r4 15 1], 0x56de
    write3(0xca5c, r4, 0xf, 0x56de);
    //0018:6      mov    [op3 0xe4ac r4 36 1], 0x40b7
    write3(0xe4ac, r4, 0x24, 0x40b7);
    //0020:3      mov    r11, [0xa000]
    r11 = *((uint16_t*)&ram[0xa000]);
    //0025:4      mov    [op3 0xeafe r4 41 1], r6
    write3(0xeafe, r4, 0x29, r6);
    //002b:5      and    [op3 0xeafe r4 41 1], r10
    write3(0xeafe, r4, 0x29, read3(0xeafe, r4, 0x29) & r10);
    //0031:6      mov    [op3 0xec86 r4 37 1], [op3 0xca5c r4 15 1]
    write3(0xec86, r4, 0x25, read3(0xca5c, r4, 0xf));
    //003a:5      and    [op3 0xec86 r4 37 1], 0x56fa
    write3(0xec86, r4, 0x25, read3(0xec86, r4, 0x25) & 0x56fa);
    //0042:2      not    [op3 0xec86 r4 37 1]
    write3(0xec86, r4, 0x25, ~read3(0xec86, r4, 0x25));
    //0047:4      mov    [op3 0xc56a r4 12 1], [op3 0xca5c r4 15 1]
    write3(0xc56a, r4, 0xc, read3(0xca5c, r4, 0xf));
    //0050:3      or     [op3 0xc56a r4 12 1], 0x56fa
    write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc) | 0x56fa);
    //0058:0      not    [op3 0xc56a r4 12 1]
    write3(0xc56a, r4, 0xc, ~read3(0xc56a, r4, 0xc));
    //005d:2      not    [op3 0xc56a r4 12 1]
    write3(0xc56a, r4, 0xc, ~read3(0xc56a, r4, 0xc));
    //0062:4      and    [op3 0xec86 r4 37 1], [op3 0xc56a r4 12 1]
    write3(0xec86, r4, 0x25, read3(0xec86, r4, 0x25) & read3(0xc56a, r4, 0xc));
    //006b:3      mov    [op3 0xca5c r4 15 1], [op3 0xec86 r4 37 1]
    write3(0xca5c, r4, 0xf, read3(0xec86, r4, 0x25));
    //0074:2      mov    [op3 0xe648 r4 40 1], r12
    write3(0xe648, r4, 0x28, r12);
    //007a:3      mov    r0, 0xfb51
    r0 = 0xfb51;
    //007f:2      mov    [op3 0xc56a r4 12 1], r11
    write3(0xc56a, r4, 0xc, r11);
    //0085:3      or     [op3 0xc56a r4 12 1], [op3 0xca5c r4 15 1]
    write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc) | read3(0xca5c, r4, 0xf));
    //008e:2      mov    [op3 0xec86 r4 37 1], r11
    write3(0xec86, r4, 0x25, r11);
    //0094:3      and    [op3 0xec86 r4 37 1], [op3 0xca5c r4 15 1]
    write3(0xec86, r4, 0x25, read3(0xec86, r4, 0x25) & read3(0xca5c, r4, 0xf));
    //009d:2      not    [op3 0xec86 r4 37 1]
    write3(0xec86, r4, 0x25, ~read3(0xec86, r4, 0x25));
    //00a2:4      and    [op3 0xc56a r4 12 1], [op3 0xec86 r4 37 1]
    write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc) & read3(0xec86, r4, 0x25));
    //00ab:3      mov    [op3 0xdf44 r4 55 1], [op3 0xc56a r4 12 1]
    write3(0xdf44, r4, 0x37, read3(0xc56a, r4, 0xc));
    //00b4:2      mov    [op3 0xeb02 r4 52 1], 0x21c7
    write3(0xeb02, r4, 0x34, 0x21c7);
    //00bb:7      shr    [op3 0xeb02 r4 52 1], 1
    write3(0xeb02, r4, 0x34, read3(0xeb02, r4, 0x34)>>1);
    //00c2:2      mov    [op3 0xf308 r4 8 1], [0xa002]
    write3(0xf308, r4, 0x8, *((uint16_t*)&ram[0xa002]));
    //00ca:1      mov    [op3 0xd67c r4 56 1], r2
```

```
    write3(0xd67c, r4, 0x38, r2);
    //00d0:2      and    [op3 0xd67c r4 56 1], r9
    write3(0xd67c, r4, 0x38, read3(0xd67c, r4, 0x38) & r9);
    //00d6:3      mov    [op3 0xc56a r4 12 1], 0x24
    write3(0xc56a, r4, 0xc, 0x24);
    //00de:0      shl    [op3 0xc56a r4 12 1], 2
    write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc)<<2);
    //00e4:3      mov    [op3 0xca5c r4 15 1], [op3 0xc56a r4 12 1]
    write3(0xca5c, r4, 0xf, read3(0xc56a, r4, 0xc));
    //00ed:2      mov    [op3 0xd1a2 r4 32 1], r0
    write3(0xd1a2, r4, 0x20, r0);
    //00f3:3      mov    [op3 0xed58 r4 53 1], 0xfc51
    write3(0xed58, r4, 0x35, 0xfc51);
    //00fb:0      not    [op3 0xed58 r4 53 1]
    write3(0xed58, r4, 0x35, ~read3(0xed58, r4, 0x35));
    //0100:2      or     [op3 0xd1a2 r4 32 1], [op3 0xed58 r4 53 1]
    write3(0xd1a2, r4, 0x20, read3(0xd1a2, r4, 0x20) | read3(0xed58, r4, 0x35));
    //0109:1      mov    [op3 0xfe22 r4 4 1], r0
    write3(0xfe22, r4, 0x4, r0);
    //010f:2      not    [op3 0xfe22 r4 4 1]
    write3(0xfe22, r4, 0x4, ~read3(0xfe22, r4, 0x4));
    //0114:4      or     [op3 0xfe22 r4 4 1], 0xfc51
    write3(0xfe22, r4, 0x4, read3(0xfe22, r4, 0x4) | 0xfc51);
    //011c:1      and    [op3 0xd1a2 r4 32 1], [op3 0xfe22 r4 4 1]
    write3(0xd1a2, r4, 0x20, read3(0xd1a2, r4, 0x20) & read3(0xfe22, r4, 0x4));
    //0125:0      not    [op3 0xd1a2 r4 32 1]
    write3(0xd1a2, r4, 0x20, ~read3(0xd1a2, r4, 0x20));
    //012a:2      mov    r0, [op3 0xd1a2 r4 32 1]
    r0 = read3(0xd1a2, r4, 0x20);
    //0130:3      mov    r8, 0x50
    r8 = 0x50;
    //0135:2      mov    [op3 0xdf44 r4 55 1], 0x0
    write3(0xdf44, r4, 0x37, 0x0);
    //013c:7      mov    [op3 0xec86 r4 37 1], 0xf2a
    write3(0xec86, r4, 0x25, 0xf2a);
    //0144:4      mov    [op3 0xdd3e r4 3 1], 0x2
    write3(0xdd3e, r4, 0x3, 0x2);
    //014c:1      mov    [op3 0xc7fc r4 54 1], [op3 0xca5c r4 15 1]
    write3(0xc7fc, r4, 0x36, read3(0xca5c, r4, 0xf));


    do
    {
        //0155:0      mov    [op3 0xfe6c r4 14 1], [op3 0xdd3e r4 3 1]
        write3(0xfe6c, r4, 0xe, read3(0xdd3e, r4, 0x3));
        //015d:7      and    [op3 0xfe6c r4 14 1], [op3 0xc7fc r4 54 1]
        write3(0xfe6c, r4, 0xe, read3(0xfe6c, r4, 0xe) & read3(0xc7fc, r4, 0x36));
        //0166:6      not    [op3 0xfe6c r4 14 1]
        write3(0xfe6c, r4, 0xe, ~read3(0xfe6c, r4, 0xe));
        //016c:0      mov    [op3 0xc56a r4 12 1], [op3 0xdd3e r4 3 1]
        write3(0xc56a, r4, 0xc, read3(0xdd3e, r4, 0x3));
        //0174:7      or     [op3 0xc56a r4 12 1], [op3 0xc7fc r4 54 1]
        write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc) | read3(0xc7fc, r4, 0x36));
        //017d:6      and    [op3 0xfe6c r4 14 1], [op3 0xc56a r4 12 1]
        write3(0xfe6c, r4, 0xe, read3(0xfe6c, r4, 0xe) & read3(0xc56a, r4, 0xc));
        //0186:5      mov    [op3 0xd1b4 r4 24 1], [op3 0xfe6c r4 14 1]
        write3(0xd1b4, r4, 0x18, read3(0xfe6c, r4, 0xe));
        //018f:4      and    [op3 0xdd3e r4 3 1], [op3 0xc7fc r4 54 1]
        write3(0xdd3e, r4, 0x3, read3(0xdd3e, r4, 0x3) & read3(0xc7fc, r4, 0x36));
        //0198:3      mov    [op3 0xc7fc r4 54 1], [op3 0xdd3e r4 3 1]
        write3(0xc7fc, r4, 0x36, read3(0xdd3e, r4, 0x3));
        //01a1:2      mov    [op3 0xdd3e r4 3 1], [op3 0xd1b4 r4 24 1]
        write3(0xdd3e, r4, 0x3, read3(0xd1b4, r4, 0x18));
        zz = read3(0xc7fc, r4, 0x36) & 0x8000;
        //01aa:1      shlf0  [op3 0xc7fc r4 54 1], 1
        write3(0xc7fc, r4, 0x36, read3(0xc7fc, r4, 0x36)<<1);
        if(zz)
        //01b0:4      orC0   [op3 0xec86 r4 37 1], 0xded2
        {write3(0xec86, r4, 0x25, read3(0xec86, r4, 0x25) | 0xded2);
        }
//write3(0xc7fc, r4, 0x36, read3(0xc7fc, r4, 0x36) & read3(0xc7fc, r4, 0x36));


    //01c1:0      jNZ0   155:0 abs
    }while(read3(0xc7fc, r4, 0x36));

    //01c6:1      not    [op3 0xec86 r4 37 1]
    write3(0xec86, r4, 0x25, ~read3(0xec86, r4, 0x25));
    //01cb:3      shlf0  [op3 0xec86 r4 37 1], 1
    write3(0xec86, r4, 0x25, read3(0xec86, r4, 0x25)<<1);
    //01d1:6      mov    [op3 0xca5c r4 15 1], [op3 0xdd3e r4 3 1]
    write3(0xca5c, r4, 0xf, read3(0xdd3e, r4, 0x3));
    //01da:5      mov    [op3 0xd1b4 r4 24 1], r13
    write3(0xd1b4, r4, 0x18, r13);
    //01e0:6      mov    [op3 0xee96 r4 45 1], r0
    write3(0xee96, r4, 0x2d, r0);
    //01e6:7      shl    [op3 0xee96 r4 45 1], 2
    write3(0xee96, r4, 0x2d, read3(0xee96, r4, 0x2d)<<2);
```

```
    //01ed:2      mov    r0, [op3 0xee96 r4 45 1]
    r0 = read3(0xee96, r4, 0x2d);

    do
    {
        //01f3:3      mov    [op3 0xed5e r4 20 1], [op3 0x9fc0 r8 7 1]
        write3(0xed5e, r4, 0x14, read3(0x9fc0, r8, 0x7));
        //01fc:2      mov    [op3 0xd3cc r4 27 1], r11
        write3(0xd3cc, r4, 0x1b, r11);
        //0202:3      mov    [op3 0xc56a r4 12 1], [op3 0xed5e r4 20 1]
        write3(0xc56a, r4, 0xc, read3(0xed5e, r4, 0x14));
        //020b:2      and    [op3 0xc56a r4 12 1], r11
        write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc) & r11);
        //0211:3      not    [op3 0xc56a r4 12 1]
        write3(0xc56a, r4, 0xc, ~read3(0xc56a, r4, 0xc));
        //0216:5      mov    [op3 0xec0e r4 5 1], [op3 0xed5e r4 20 1]
        write3(0xec0e, r4, 0x5, read3(0xed5e, r4, 0x14));
        //021f:4      or     [op3 0xec0e r4 5 1], r11
        write3(0xec0e, r4, 0x5, read3(0xec0e, r4, 0x5) | r11);
        //0225:5      and    [op3 0xc56a r4 12 1], [op3 0xec0e r4 5 1]
        write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc) & read3(0xec0e, r4, 0x5));
        //022e:4      mov    1[0x9fc0 + r8], [op3 0xc56a r4 12 1]
        *((uint16_t*)&ram[0x9fc0 + r8]) = read3(0xc56a, r4, 0xc);
        //0236:7      mov    [op3 0xd260 r4 56 1], 0x1bda
        write3(0xd260, r4, 0x38, 0x1bda);
        //023e:4      mov    [op3 0xea0a r4 51 1], [op3 0xdf44 r4 55 1]
        write3(0xea0a, r4, 0x33, read3(0xdf44, r4, 0x37));
        //0247:3      mov    [op3 0xedce r4 45 1], r11
        write3(0xedce, r4, 0x2d, r11);
        do
        {
            //024d:4      mov    [op3 0xc56a r4 12 1], [op3 0xea0a r4 51 1]
            write3(0xc56a, r4, 0xc, read3(0xea0a, r4, 0x33));
            //0256:3      not    [op3 0xc56a r4 12 1]
            write3(0xc56a, r4, 0xc, ~read3(0xc56a, r4, 0xc));
            //025b:5      and    [op3 0xc56a r4 12 1], [op3 0xedce r4 45 1]
            write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc) & read3(0xedce, r4, 0x2d));
            //0264:4      mov    [op3 0xfa8c r4 9 1], [op3 0xea0a r4 51 1]
            write3(0xfa8c, r4, 0x9, read3(0xea0a, r4, 0x33));
            //026d:3      mov    [op3 0xfb56 r4 4 1], [op3 0xedce r4 45 1]
            write3(0xfb56, r4, 0x4, read3(0xedce, r4, 0x2d));
            //0276:2      not    [op3 0xfb56 r4 4 1]
            write3(0xfb56, r4, 0x4, ~read3(0xfb56, r4, 0x4));
            //027b:4      and    [op3 0xfa8c r4 9 1], [op3 0xfb56 r4 4 1]
            write3(0xfa8c, r4, 0x9, read3(0xfa8c, r4, 0x9) & read3(0xfb56, r4, 0x4));
            //0284:3      or     [op3 0xc56a r4 12 1], [op3 0xfa8c r4 9 1]
            write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc) | read3(0xfa8c, r4, 0x9));
            //028d:2      mov    [op3 0xc5c0 r4 62 1], [op3 0xc56a r4 12 1]
            write3(0xc5c0, r4, 0x3e, read3(0xc56a, r4, 0xc));
            //0296:1      and    [op3 0xea0a r4 51 1], [op3 0xedce r4 45 1]
            write3(0xea0a, r4, 0x33, read3(0xea0a, r4, 0x33) & read3(0xedce, r4, 0x2d));
            //029f:0      mov    [op3 0xedce r4 45 1], [op3 0xea0a r4 51 1]
            write3(0xedce, r4, 0x2d, read3(0xea0a, r4, 0x33));
            //02a7:7      mov    [op3 0xea0a r4 51 1], [op3 0xc5c0 r4 62 1]
            write3(0xea0a, r4, 0x33, read3(0xc5c0, r4, 0x3e));
            zz = read3(0xedce, r4, 0x2d) & 0x8000;
            //02b0:6      shlf0  [op3 0xedce r4 45 1], 1
            write3(0xedce, r4, 0x2d, read3(0xedce, r4, 0x2d) <<1);
            if(zz){
            //02b7:1      orC0   [op3 0xd260 r4 56 1], 0xd3dc
            write3(0xd260, r4, 0x38, read3(0xd260, r4, 0x38) | 0xd3dc);
            }
//write3(0xedce, r4, 0x2d, read3(0xedce, r4, 0x2d) & read3(0xedce, r4, 0x2d));

        //02c7:5      jNZ0   24d:4 abs
        }while(read3(0xedce, r4, 0x2d));

        //02cc:6      not    [op3 0xd260 r4 56 1]
        write3(0xd260, r4, 0x38, ~read3(0xd260, r4, 0x38));
        //02d2:0      shlf0  [op3 0xd260 r4 56 1], 1
        write3(0xd260, r4, 0x38, read3(0xd260, r4, 0x38)<<1);
        //02d8:3      mov    r11, [op3 0xea0a r4 51 1]
        r11 = read3(0xea0a, r4, 0x33);
        //02de:4      mov    [op3 0xec0e r4 5 1], r6
        write3(0xec0e, r4, 0x5, r6);
        //02e4:5      mov    [op3 0xec86 r4 37 1], 0x4d6c
        write3(0xec86, r4, 0x25, 0x4d6c);
        //02ec:2      mov    [op3 0xc5de r4 36 1], 0x2
        write3(0xc5de, r4, 0x24, 0x2);
        //02f3:7      mov    [op3 0xf6a2 r4 57 1], r8
        write3(0xf6a2, r4, 0x39, r8);
        do
        {
            //02fa:0      mov    [op3 0xc812 r4 19 1], [op3 0xc5de r4 36 1]
            write3(0xc812, r4, 0x13, read3(0xc5de, r4, 0x24));
            //0302:7      not    [op3 0xc812 r4 19 1]
            write3(0xc812, r4, 0x13, ~read3(0xc812, r4, 0x13));
            //0308:1      and    [op3 0xc812 r4 19 1], [op3 0xf6a2 r4 57 1]
```

```
            write3(0xc812, r4, 0x13, read3(0xc812, r4, 0x13) & read3(0xf6a2, r4, 0x39));
            //0311:0      mov    [op3 0xee1a r4 54 1], [op3 0xc5de r4 36 1]
            write3(0xee1a, r4, 0x36, read3(0xc5de, r4, 0x24));
            //0319:7      not    [op3 0xee1a r4 54 1]
            write3(0xee1a, r4, 0x36, ~read3(0xee1a, r4, 0x36));
            //031f:1      or     [op3 0xee1a r4 54 1], [op3 0xf6a2 r4 57 1]
            write3(0xee1a, r4, 0x36, read3(0xee1a, r4, 0x36) | read3(0xf6a2, r4, 0x39));
            //0328:0      not    [op3 0xee1a r4 54 1]
            write3(0xee1a, r4, 0x36, ~read3(0xee1a, r4, 0x36));
            //032d:2      or     [op3 0xc812 r4 19 1], [op3 0xee1a r4 54 1]
            write3(0xc812, r4, 0x13, read3(0xc812, r4, 0x13) | read3(0xee1a, r4, 0x36));
            //0336:1      mov    [op3 0xecac r4 33 1], [op3 0xc812 r4 19 1]
            write3(0xecac, r4, 0x21, read3(0xc812, r4, 0x13));
            //033f:0      and    [op3 0xc5de r4 36 1], [op3 0xf6a2 r4 57 1]
            write3(0xc5de, r4, 0x24, read3(0xc5de, r4, 0x24) & read3(0xf6a2, r4, 0x39));
            //0347:7      mov    [op3 0xf6a2 r4 57 1], [op3 0xc5de r4 36 1]
            write3(0xf6a2, r4, 0x39, read3(0xc5de, r4, 0x24));
            //0350:6      mov    [op3 0xc5de r4 36 1], [op3 0xecac r4 33 1]
            write3(0xc5de, r4, 0x24, read3(0xecac, r4, 0x21));
            zz = read3(0xf6a2, r4, 0x39) & 0x8000;
            //0359:5      shlf1  [op3 0xf6a2 r4 57 1], 1
            write3(0xf6a2, r4, 0x39, read3(0xf6a2, r4, 0x39)<<1);
            if(zz){
            //0360:0      orC1   [op3 0xec86 r4 37 1], 0x99c3
            write3(0xec86, r4, 0x25, read3(0xec86, r4, 0x25) | 0x99c3);
            }
//write3(0xf6a2, r4, 0x39, read3(0xf6a2, r4, 0x39) & read3(0xf6a2, r4, 0x39));

        //0370:4      jNZ1   2fa:0 abs
        }while(read3(0xf6a2, r4, 0x39));

        //0375:5      not    [op3 0xec86 r4 37 1]
        write3(0xec86, r4, 0x25, ~read3(0xec86, r4, 0x25));
        //037a:7      shlf1  [op3 0xec86 r4 37 1], 1
        write3(0xec86, r4, 0x25, read3(0xec86, r4, 0x25)<<1);
        //0381:2      mov    r8, [op3 0xc5de r4 36 1]
        r8 = read3(0xc5de, r4, 0x24);
        //0387:3      mov    [op3 0xf5d6 r4 50 1], [op3 0x9fc0 r8 7 1]
        write3(0xf5d6, r4, 0x32, read3(0x9fc0, r8, 0x7));
        //0390:2      mov    [op3 0xec86 r4 37 1], [op3 0xf5d6 r4 50 1]
        write3(0xec86, r4, 0x25, read3(0xf5d6, r4, 0x32));
        //0399:1      and    [op3 0xec86 r4 37 1], [op3 0xf308 r4 8 1]
        write3(0xec86, r4, 0x25, read3(0xec86, r4, 0x25) & read3(0xf308, r4, 0x8));
        //03a2:0      mov    [op3 0xc56a r4 12 1], [op3 0xf5d6 r4 50 1]
        write3(0xc56a, r4, 0xc, read3(0xf5d6, r4, 0x32));
        //03aa:7      or     [op3 0xc56a r4 12 1], [op3 0xf308 r4 8 1]
        write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc) | read3(0xf308, r4, 0x8));
        //03b3:6      not    [op3 0xc56a r4 12 1]
        write3(0xc56a, r4, 0xc, ~read3(0xc56a, r4, 0xc));
        //03b9:0      or     [op3 0xec86 r4 37 1], [op3 0xc56a r4 12 1]
        write3(0xec86, r4, 0x25, read3(0xec86, r4, 0x25) | read3(0xc56a, r4, 0xc));
        //03c1:7      not    [op3 0xec86 r4 37 1]
        write3(0xec86, r4, 0x25, ~read3(0xec86, r4, 0x25));
        //03c7:1      mov    1[0x9fc0 + r8], [op3 0xec86 r4 37 1]
        *((uint16_t*)&ram[0x9fc0 + r8]) = read3(0xec86, r4, 0x25);
        //03cf:4      mov    [op3 0xec86 r4 37 1], 0x5ef
        write3(0xec86, r4, 0x25, 0x5ef);
        //03d7:1      shr    [op3 0xec86 r4 37 1], 1
        write3(0xec86, r4, 0x25, read3(0xec86, r4, 0x25)>>1);
        //03dd:4      mov    [op3 0xec86 r4 37 1], [op3 0xdf44 r4 55 1]
        write3(0xec86, r4, 0x25, read3(0xdf44, r4, 0x37));
        //03e6:3      not    [op3 0xec86 r4 37 1]
        write3(0xec86, r4, 0x25, ~read3(0xec86, r4, 0x25));
        //03eb:5      mov    [op3 0xde48 r4 30 1], 0x1b51
        write3(0xde48, r4, 0x1e, 0x1b51);
        //03f3:2      mov    [op3 0xd54a r4 9 1], 0x1
        write3(0xd54a, r4, 0x9, 0x1);
        //03fa:7      mov    [op3 0xea56 r4 60 1], [op3 0xec86 r4 37 1]
        write3(0xea56, r4, 0x3c, read3(0xec86, r4, 0x25));
        do
        {
            //0403:6      mov    [op3 0xcbbe r4 2 1], [op3 0xd54a r4 9 1]
            write3(0xcbbe, r4, 0x2, read3(0xd54a, r4, 0x9));
            //040c:5      or     [op3 0xcbbe r4 2 1], [op3 0xea56 r4 60 1]
            write3(0xcbbe, r4, 0x2, read3(0xcbbe, r4, 0x2) | read3(0xea56, r4, 0x3c));
            //0415:4      mov    [op3 0xeaee r4 36 1], [op3 0xd54a r4 9 1]
            write3(0xeaee, r4, 0x24, read3(0xd54a, r4, 0x9));
            //041e:3      and    [op3 0xeaee r4 36 1], [op3 0xea56 r4 60 1]
            write3(0xeaee, r4, 0x24, read3(0xeaee, r4, 0x24) & read3(0xea56, r4, 0x3c));
            //0427:2      not    [op3 0xeaee r4 36 1]
            write3(0xeaee, r4, 0x24, ~read3(0xeaee, r4, 0x24));
            //042c:4      and    [op3 0xcbbe r4 2 1], [op3 0xeaee r4 36 1]
            write3(0xcbbe, r4, 0x2, read3(0xcbbe, r4, 0x2) & read3(0xeaee, r4, 0x24));
            //0435:3      mov    [op3 0xc56a r4 12 1], [op3 0xcbbe r4 2 1]
            write3(0xc56a, r4, 0xc, read3(0xcbbe, r4, 0x2));
            //043e:2      and    [op3 0xd54a r4 9 1], [op3 0xea56 r4 60 1]
            write3(0xd54a, r4, 0x9, read3(0xd54a, r4, 0x9) & read3(0xea56, r4, 0x3c));
            //0447:1      mov    [op3 0xea56 r4 60 1], [op3 0xd54a r4 9 1]
```

```
            write3(0xea56, r4, 0x3c, read3(0xd54a, r4, 0x9));
            //0450:0      mov    [op3 0xd54a r4 9 1], [op3 0xc56a r4 12 1]
            write3(0xd54a, r4, 0x9, read3(0xc56a, r4, 0xc));
            zz = read3(0xea56, r4, 0x3c) & 0x8000;
            //0458:7      shlf0  [op3 0xea56 r4 60 1], 1
            write3(0xea56, r4, 0x3c, read3(0xea56, r4, 0x3c)<<1);
            if(zz){
            //045f:2      orC0   [op3 0xde48 r4 30 1], 0xfd25
            write3(0xde48, r4, 0x1e, read3(0xde48, r4, 0x1e) | 0xfd25);
            }
//write3(0xea56, r4, 0x3c, read3(0xea56, r4, 0x3c) & read3(0xea56, r4, 0x3c));

        //046f:6      jNZ0   403:6 abs
        }while(read3(0xea56, r4, 0x3c));

        //0474:7      not    [op3 0xde48 r4 30 1]
        write3(0xde48, r4, 0x1e, ~read3(0xde48, r4, 0x1e));
        //047a:1      shlf0  [op3 0xde48 r4 30 1], 1
        write3(0xde48, r4, 0x1e, read3(0xde48, r4, 0x1e)<<1);
        //0480:4      mov    [op3 0xec86 r4 37 1], [op3 0xd54a r4 9 1]
        write3(0xec86, r4, 0x25, read3(0xd54a, r4, 0x9));
        //0489:3      mov    [op3 0xc56a r4 12 1], 0x69f5
        write3(0xc56a, r4, 0xc, 0x69f5);
        //0491:0      mov    [op3 0xcb60 r4 43 1], [op3 0xf308 r4 8 1]
        write3(0xcb60, r4, 0x2b, read3(0xf308, r4, 0x8));
        //0499:7      mov    [op3 0xcac2 r4 41 1], [op3 0xec86 r4 37 1]
        write3(0xcac2, r4, 0x29, read3(0xec86, r4, 0x25));
        do
        {
            //04a2:6      mov    [op3 0xdb1e r4 33 1], [op3 0xcb60 r4 43 1]
            write3(0xdb1e, r4, 0x21, read3(0xcb60, r4, 0x2b));
            //04ab:5      or     [op3 0xdb1e r4 33 1], [op3 0xcac2 r4 41 1]
            write3(0xdb1e, r4, 0x21, read3(0xdb1e, r4, 0x21) | read3(0xcac2, r4, 0x29));
            //04b4:4      mov    [op3 0xdc92 r4 43 1], [op3 0xcb60 r4 43 1]
            write3(0xdc92, r4, 0x2b, read3(0xcb60, r4, 0x2b));
            //04bd:3      and    [op3 0xdc92 r4 43 1], [op3 0xcac2 r4 41 1]
            write3(0xdc92, r4, 0x2b, read3(0xdc92, r4, 0x2b) & read3(0xcac2, r4, 0x29));
            //04c6:2      not    [op3 0xdc92 r4 43 1]
            write3(0xdc92, r4, 0x2b, ~read3(0xdc92, r4, 0x2b));
            //04cb:4      and    [op3 0xdb1e r4 33 1], [op3 0xdc92 r4 43 1]
            write3(0xdb1e, r4, 0x21, read3(0xdb1e, r4, 0x21) & read3(0xdc92, r4, 0x2b));
            //04d4:3      mov    [op3 0xd31c r4 19 1], [op3 0xdb1e r4 33 1]
            write3(0xd31c, r4, 0x13, read3(0xdb1e, r4, 0x21));
            //04dd:2      and    [op3 0xcb60 r4 43 1], [op3 0xcac2 r4 41 1]
            write3(0xcb60, r4, 0x2b, read3(0xcb60, r4, 0x2b) & read3(0xcac2, r4, 0x29));
            //04e6:1      mov    [op3 0xcac2 r4 41 1], [op3 0xcb60 r4 43 1]
            write3(0xcac2, r4, 0x29, read3(0xcb60, r4, 0x2b));
            //04ef:0      mov    [op3 0xcb60 r4 43 1], [op3 0xd31c r4 19 1]
            write3(0xcb60, r4, 0x2b, read3(0xd31c, r4, 0x13));
            zz = read3(0xcac2, r4, 0x29) & 0x8000;
            //04f7:7      shlf1  [op3 0xcac2 r4 41 1], 1
            write3(0xcac2, r4, 0x29, read3(0xcac2, r4, 0x29)<<1);
            if(zz){
            //04fe:2      orC1   [op3 0xc56a r4 12 1], 0xeda8
            write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc) | 0xeda8);
            //write3(0xcac2, r4, 0x29, read3(0xcac2, r4, 0x29) & read3(0xcac2, r4, 0x29));
            }
        //050e:6      jNZ1   4a2:6 abs
        }while(read3(0xcac2, r4, 0x29));

        //0513:7      not    [op3 0xc56a r4 12 1]
        write3(0xc56a, r4, 0xc, ~read3(0xc56a, r4, 0xc));
        //0519:1      shlf1  [op3 0xc56a r4 12 1], 1
        write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc)<<1);
        //051f:4      mov    [op3 0xf308 r4 8 1], [op3 0xcb60 r4 43 1]
        write3(0xf308, r4, 0x8, read3(0xcb60, r4, 0x2b));
        //0528:3      mov    [op3 0xec86 r4 37 1], 0x6030
        write3(0xec86, r4, 0x25, 0x6030);
        //0530:0      mov    [op3 0xc474 r4 61 1], 0x53e8
        write3(0xc474, r4, 0x3d, 0x53e8);
        //0537:5      mov    [op3 0xc802 r4 15 1], 0x2
        write3(0xc802, r4, 0xf, 0x2);
        //053f:2      mov    [op3 0xe69c r4 15 1], r8
        write3(0xe69c, r4, 0xf, r8);
        do
        {
            //0545:3      mov    [op3 0xec86 r4 37 1], [op3 0xc802 r4 15 1]
            write3(0xec86, r4, 0x25, read3(0xc802, r4, 0xf));
            //054e:2      and    [op3 0xec86 r4 37 1], [op3 0xe69c r4 15 1]
            write3(0xec86, r4, 0x25, read3(0xec86, r4, 0x25) & read3(0xe69c, r4, 0xf));
            //0557:1      not    [op3 0xec86 r4 37 1]
            write3(0xec86, r4, 0x25, ~read3(0xec86, r4, 0x25));
            //055c:3      mov    [op3 0xd556 r4 6 1], [op3 0xc802 r4 15 1]
            write3(0xd556, r4, 0x6, read3(0xc802, r4, 0xf));
            //0565:2      or     [op3 0xd556 r4 6 1], [op3 0xe69c r4 15 1]
            write3(0xd556, r4, 0x6, read3(0xd556, r4, 0x6) | read3(0xe69c, r4, 0xf));
            //056e:1      and    [op3 0xec86 r4 37 1], [op3 0xd556 r4 6 1]
            write3(0xec86, r4, 0x25, read3(0xec86, r4, 0x25) & read3(0xd556, r4, 0x6));
```

```
            //0577:0      mov    [op3 0xee0c r4 39 1], [op3 0xec86 r4 37 1]
            write3(0xee0c, r4, 0x27, read3(0xec86, r4, 0x25));
            //057f:7      and    [op3 0xc802 r4 15 1], [op3 0xe69c r4 15 1]
            write3(0xc802, r4, 0xf, read3(0xc802, r4, 0xf) & read3(0xe69c, r4, 0xf));
            //0588:6      mov    [op3 0xe69c r4 15 1], [op3 0xc802 r4 15 1]
            write3(0xe69c, r4, 0xf, read3(0xc802, r4, 0xf));
            //0591:5      mov    [op3 0xc802 r4 15 1], [op3 0xee0c r4 39 1]
            write3(0xc802, r4, 0xf, read3(0xee0c, r4, 0x27));
            zz = read3(0xe69c, r4, 0xf) & 0x8000;
            //059a:4      shlf1  [op3 0xe69c r4 15 1], 1
            write3(0xe69c, r4, 0xf, read3(0xe69c, r4, 0xf)<<1);
            if(zz){
            //05a0:7      orC1   [op3 0xc474 r4 61 1], 0xd1c2
            write3(0xc474, r4, 0x3d, read3(0xc474, r4, 0x3d) | 0xd1c2);
            //write3(0xe69c, r4, 0xf, read3(0xe69c, r4, 0xf) & read3(0xe69c, r4, 0xf));
            }
        //05b1:3      jNZ1   545:3 abs
        }while(read3(0xe69c, r4, 0xf));


        //05b6:4      not    [op3 0xc474 r4 61 1]
        write3(0xc474, r4, 0x3d, ~read3(0xc474, r4, 0x3d));
        //05bb:6      shlf1  [op3 0xc474 r4 61 1], 1
        write3(0xc474, r4, 0x3d, read3(0xc474, r4, 0x3d)<<1);
        //05c2:1      mov    r8, [op3 0xc802 r4 15 1]
        r8 = read3(0xc802, r4, 0xf);
        //05c8:2      mov    [op3 0xc56a r4 12 1], 0x2c71
        write3(0xc56a, r4, 0xc, 0x2c71);
        //05cf:7      mov    [op3 0xcf66 r4 63 1], 0x1
        write3(0xcf66, r4, 0x3f, 0x1);
        //05d7:4      mov    [op3 0xdb5e r4 33 1], [op3 0xdf44 r4 55 1]
        write3(0xdb5e, r4, 0x21, read3(0xdf44, r4, 0x37));
        do
        {
            //05e0:3      mov    [op3 0xc7e0 r4 57 1], [op3 0xcf66 r4 63 1]
            write3(0xc7e0, r4, 0x39, read3(0xcf66, r4, 0x3f));
            //05e9:2      mov    [op3 0xf7a4 r4 59 1], [op3 0xdb5e r4 33 1]
            write3(0xf7a4, r4, 0x3b, read3(0xdb5e, r4, 0x21));
            //05f2:1      not    [op3 0xf7a4 r4 59 1]
            write3(0xf7a4, r4, 0x3b, ~read3(0xf7a4, r4, 0x3b));
            //05f7:3      or     [op3 0xc7e0 r4 57 1], [op3 0xf7a4 r4 59 1]
            write3(0xc7e0, r4, 0x39, read3(0xc7e0, r4, 0x39) | read3(0xf7a4, r4, 0x3b));
            //0600:2      not    [op3 0xc7e0 r4 57 1]
            write3(0xc7e0, r4, 0x39, ~read3(0xc7e0, r4, 0x39));
            //0605:4      mov    [op3 0xdade r4 48 1], [op3 0xcf66 r4 63 1]
            write3(0xdade, r4, 0x30, read3(0xcf66, r4, 0x3f));
            //060e:3      mov    [op3 0xc6f4 r4 32 1], [op3 0xdb5e r4 33 1]
            write3(0xc6f4, r4, 0x20, read3(0xdb5e, r4, 0x21));
            //0617:2      not    [op3 0xc6f4 r4 32 1]
            write3(0xc6f4, r4, 0x20, ~read3(0xc6f4, r4, 0x20));
            //061c:4      and    [op3 0xdade r4 48 1], [op3 0xc6f4 r4 32 1]
            write3(0xdade, r4, 0x30, read3(0xdade, r4, 0x30) & read3(0xc6f4, r4, 0x20));
            //0625:3      or     [op3 0xc7e0 r4 57 1], [op3 0xdade r4 48 1]
            write3(0xc7e0, r4, 0x39, read3(0xc7e0, r4, 0x39) | read3(0xdade, r4, 0x30));
            //062e:2      mov    [op3 0xefce r4 55 1], [op3 0xc7e0 r4 57 1]
            write3(0xefce, r4, 0x37, read3(0xc7e0, r4, 0x39));
            //0637:1      and    [op3 0xcf66 r4 63 1], [op3 0xdb5e r4 33 1]
            write3(0xcf66, r4, 0x3f, read3(0xcf66, r4, 0x3f) & read3(0xdb5e, r4, 0x21));
            //0640:0      mov    [op3 0xdb5e r4 33 1], [op3 0xcf66 r4 63 1]
            write3(0xdb5e, r4, 0x21, read3(0xcf66, r4, 0x3f));
            //0648:7      mov    [op3 0xcf66 r4 63 1], [op3 0xefce r4 55 1]
            write3(0xcf66, r4, 0x3f, read3(0xefce, r4, 0x37));
            zz = read3(0xdb5e, r4, 0x21);
            //0651:6      shlf0  [op3 0xdb5e r4 33 1], 1
            write3(0xdb5e, r4, 0x21, read3(0xdb5e, r4, 0x21)<<1);
            if(zz)
            //0658:1      orC0   [op3 0xc56a r4 12 1], 0xcddc
            {write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc) | 0xcddc);
            //write3(0xdb5e, r4, 0x21, read3(0xdb5e, r4, 0x21) & read3(0xdb5e, r4, 0x21));
            }
        //0668:5      jNZ0   5e0:3 abs
        }while(read3(0xdb5e, r4, 0x21));

        //066d:6      not    [op3 0xc56a r4 12 1]
        write3(0xc56a, r4, 0xc, ~read3(0xc56a, r4, 0xc));
        //0673:0      shlf0  [op3 0xc56a r4 12 1], 1
        write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc)<<1);
        //0679:3      mov    [op3 0xdf44 r4 55 1], [op3 0xcf66 r4 63 1]
        write3(0xdf44, r4, 0x37, read3(0xcf66, r4, 0x3f));
        //0682:2      mov    [op3 0xcf66 r4 63 1], [op3 0xdf44 r4 55 1]
        write3(0xcf66, r4, 0x3f, read3(0xdf44, r4, 0x37));
        //068b:1      mov    [op3 0xc56a r4 12 1], 0x40
        write3(0xc56a, r4, 0xc, 0x40);
        //0692:6      not    [op3 0xc56a r4 12 1]
        write3(0xc56a, r4, 0xc, ~read3(0xc56a, r4, 0xc));
        //0698:0      mov    [op3 0xe740 r4 42 1], 0x4c6c
        write3(0xe740, r4, 0x2a, 0x4c6c);
        //069f:5      mov    [op3 0xcaf8 r4 21 1], 0x1
```

```
        write3(0xcaf8, r4, 0x15, 0x1);
        //06a7:2      mov    [op3 0xce44 r4 61 1], [op3 0xc56a r4 12 1]
        write3(0xce44, r4, 0x3d, read3(0xc56a, r4, 0xc));
        do
        {
            //06b0:1      mov    [op3 0xe242 r4 42 1], [op3 0xcaf8 r4 21 1]
            write3(0xe242, r4, 0x2a, read3(0xcaf8, r4, 0x15));
            //06b9:0      or     [op3 0xe242 r4 42 1], [op3 0xce44 r4 61 1]
            write3(0xe242, r4, 0x2a, read3(0xe242, r4, 0x2a) | read3(0xce44, r4, 0x3d));
            //06c1:7      mov    [op3 0xdfa0 r4 8 1], [op3 0xcaf8 r4 21 1]
            write3(0xdfa0, r4, 0x8, read3(0xcaf8, r4, 0x15));
            //06ca:6      and    [op3 0xdfa0 r4 8 1], [op3 0xce44 r4 61 1]
            write3(0xdfa0, r4, 0x8, read3(0xdfa0, r4, 0x8) & read3(0xce44, r4, 0x3d));
            //06d3:5      not    [op3 0xdfa0 r4 8 1]
            write3(0xdfa0, r4, 0x8, ~read3(0xdfa0, r4, 0x8));
            //06d8:7      and    [op3 0xe242 r4 42 1], [op3 0xdfa0 r4 8 1]
            write3(0xe242, r4, 0x2a, read3(0xe242, r4, 0x2a) & read3(0xdfa0, r4, 0x8));
            //06e1:6      mov    [op3 0xe4c8 r4 39 1], [op3 0xe242 r4 42 1]
            write3(0xe4c8, r4, 0x27, read3(0xe242, r4, 0x2a));
            //06ea:5      and    [op3 0xcaf8 r4 21 1], [op3 0xce44 r4 61 1]
            write3(0xcaf8, r4, 0x15, read3(0xcaf8, r4, 0x15) & read3(0xce44, r4, 0x3d));
            //06f3:4      mov    [op3 0xce44 r4 61 1], [op3 0xcaf8 r4 21 1]
            write3(0xce44, r4, 0x3d, read3(0xcaf8, r4, 0x15));
            //06fc:3      mov    [op3 0xcaf8 r4 21 1], [op3 0xe4c8 r4 39 1]
            write3(0xcaf8, r4, 0x15, read3(0xe4c8, r4, 0x27));
            zz = read3(0xce44, r4, 0x3d) & 0x8000;
            //0705:2      shlf1  [op3 0xce44 r4 61 1], 1
            write3(0xce44, r4, 0x3d, read3(0xce44, r4, 0x3d)<<1);
            if(zz)
            //070b:5      orC1   [op3 0xe740 r4 42 1], 0x8127
            {write3(0xe740, r4, 0x2a, read3(0xe740, r4, 0x2a) | 0x8127);
            //write3(0xce44, r4, 0x3d, read3(0xce44, r4, 0x3d) & read3(0xce44, r4, 0x3d));
            }
        //071c:1      jNZ1   6b0:1 abs
        }while(read3(0xce44, r4, 0x3d));


        //0721:2      not    [op3 0xe740 r4 42 1]
        write3(0xe740, r4, 0x2a, ~read3(0xe740, r4, 0x2a));
        //0726:4      shlf1  [op3 0xe740 r4 42 1], 1
        write3(0xe740, r4, 0x2a, read3(0xe740, r4, 0x2a)<<1);
        //072c:7      mov    [op3 0xc56a r4 12 1], [op3 0xcaf8 r4 21 1]
        write3(0xc56a, r4, 0xc, read3(0xcaf8, r4, 0x15));
        //0735:6      mov    [op3 0xd0d2 r4 5 1], 0x28e1
        write3(0xd0d2, r4, 0x5, 0x28e1);
        //073d:3      mov    [op3 0xc6f2 r4 24 1], [op3 0xcf66 r4 63 1]
        write3(0xc6f2, r4, 0x18, read3(0xcf66, r4, 0x3f));
        //0746:2      mov    [op3 0xc2d6 r4 44 1], [op3 0xc56a r4 12 1]
        write3(0xc2d6, r4, 0x2c, read3(0xc56a, r4, 0xc));
        do
        {
            //074f:1      mov    [op3 0xe074 r4 17 1], [op3 0xc6f2 r4 24 1]
            write3(0xe074, r4, 0x11, read3(0xc6f2, r4, 0x18));
            //0758:0      not    [op3 0xe074 r4 17 1]
            write3(0xe074, r4, 0x11, ~read3(0xe074, r4, 0x11));
            //075d:2      or     [op3 0xe074 r4 17 1], [op3 0xc2d6 r4 44 1]
            write3(0xe074, r4, 0x11, read3(0xe074, r4, 0x11) | read3(0xc2d6, r4, 0x2c));
            //0766:1      mov    [op3 0xc086 r4 45 1], [op3 0xc6f2 r4 24 1]
            write3(0xc086, r4, 0x2d, read3(0xc6f2, r4, 0x18));
            //076f:0      mov    [op3 0xe356 r4 24 1], [op3 0xc2d6 r4 44 1]
            write3(0xe356, r4, 0x18, read3(0xc2d6, r4, 0x2c));
            //0777:7      not    [op3 0xe356 r4 24 1]
            write3(0xe356, r4, 0x18, ~read3(0xe356, r4, 0x18));
            //077d:1      or     [op3 0xc086 r4 45 1], [op3 0xe356 r4 24 1]
            write3(0xc086, r4, 0x2d, read3(0xc086, r4, 0x2d) | read3(0xe356, r4, 0x18));
            //0786:0      and    [op3 0xe074 r4 17 1], [op3 0xc086 r4 45 1]
            write3(0xe074, r4, 0x11, read3(0xe074, r4, 0x11) & read3(0xc086, r4, 0x2d));
            //078e:7      not    [op3 0xe074 r4 17 1]
            write3(0xe074, r4, 0x11, ~read3(0xe074, r4, 0x11));
            //0794:1      mov    [op3 0xec86 r4 37 1], [op3 0xe074 r4 17 1]
            write3(0xec86, r4, 0x25, read3(0xe074, r4, 0x11));
            //079d:0      and    [op3 0xc6f2 r4 24 1], [op3 0xc2d6 r4 44 1]
            write3(0xc6f2, r4, 0x18, read3(0xc6f2, r4, 0x18) & read3(0xc2d6, r4, 0x2c));
            //07a5:7      mov    [op3 0xc2d6 r4 44 1], [op3 0xc6f2 r4 24 1]
            write3(0xc2d6, r4, 0x2c, read3(0xc6f2, r4, 0x18));
            //07ae:6      mov    [op3 0xc6f2 r4 24 1], [op3 0xec86 r4 37 1]
            write3(0xc6f2, r4, 0x18, read3(0xec86, r4, 0x25));
            zz = read3(0xc2d6, r4, 0x2c) & 0x8000;
            //07b7:5      shlf1  [op3 0xc2d6 r4 44 1], 1
            write3(0xc2d6, r4, 0x2c, read3(0xc2d6, r4, 0x2c)<<1);
            if(zz){
            //07be:0      orC1   [op3 0xd0d2 r4 5 1], 0xef91
            write3(0xd0d2, r4, 0x5, read3(0xd0d2, r4, 0x5) | 0xef91);
            }
//write3(0xc2d6, r4, 0x2c, read3(0xc2d6, r4, 0x2c) & read3(0xc2d6, r4, 0x2c));

        //07ce:4      jNZ1   74f:1 abs
        }while(read3(0xc2d6, r4, 0x2c));
```

```
693.
694.         //07d3:5      not    [op3 0xd0d2 r4 5 1]
695.         write3(0xd0d2, r4, 0x5, ~read3(0xd0d2, r4, 0x5));
696.         //07d8:7      shlf1  [op3 0xd0d2 r4 5 1], 1
697.         write3(0xd0d2, r4, 0x5, read3(0xd0d2, r4, 0x5)<<1);
698.         //07df:2      mov    [op3 0xcf66 r4 63 1], [op3 0xc6f2 r4 24 1]
699.         write3(0xcf66, r4, 0x3f, read3(0xc6f2, r4, 0x18));
700. //write3(0xcf66, r4, 0x3f, read3(0xcf66, r4, 0x3f) & read3(0xcf66, r4, 0x3f));
701.
702.
703.         //07f1:0      jNZ1   c60:1 abs
704.         if(!read3(0xcf66, r4, 0x3f))
705.         {
706.             //07f6:1      mov    [op3 0xcf66 r4 63 1], 0x1fbf
707.             write3(0xcf66, r4, 0x3f, 0x1fbf);
708.             //07fd:6      mov    [op3 0xec86 r4 37 1], 0xb000
709.             write3(0xec86, r4, 0x25, 0xb000);
710.             //0805:3      mov    [op3 0xe968 r4 35 1], 0x5f32
711.             write3(0xe968, r4, 0x23, 0x5f32);
712.             //080d:0      shr    [op3 0xe968 r4 35 1], 1
713.             write3(0xe968, r4, 0x23, read3(0xe968, r4, 0x23)>>1);
714.             //0813:3      mov    [op3 0xc56a r4 12 1], 0x2731
715.             write3(0xc56a, r4, 0xc, 0x2731);
716.             //081b:0      mov    [op3 0xe0b8 r4 18 1], r0
717.             write3(0xe0b8, r4, 0x12, r0);
718.             //0821:1      mov    [op3 0xc56e r4 57 1], [op3 0xec86 r4 37 1]
719.             write3(0xc56e, r4, 0x39, read3(0xec86, r4, 0x25));
720.             do
721.             {
722.                 //082a:0      mov    [op3 0xf142 r4 35 1], [op3 0xe0b8 r4 18 1]
723.                 write3(0xf142, r4, 0x23, read3(0xe0b8, r4, 0x12));
724.                 //0832:7      and    [op3 0xf142 r4 35 1], [op3 0xc56e r4 57 1]
725.                 write3(0xf142, r4, 0x23, read3(0xf142, r4, 0x23) & read3(0xc56e, r4, 0x39));
726.                 //083b:6      not    [op3 0xf142 r4 35 1]
727.                 write3(0xf142, r4, 0x23, ~read3(0xf142, r4, 0x23));
728.                 //0841:0      mov    [op3 0xc10e r4 55 1], [op3 0xe0b8 r4 18 1]
729.                 write3(0xc10e, r4, 0x37, read3(0xe0b8, r4, 0x12));
730.                 //0849:7      or     [op3 0xc10e r4 55 1], [op3 0xc56e r4 57 1]
731.                 write3(0xc10e, r4, 0x37, read3(0xc10e, r4, 0x37) | read3(0xc56e, r4, 0x39));
732.                 //0852:6      not    [op3 0xc10e r4 55 1]
733.                 write3(0xc10e, r4, 0x37, ~read3(0xc10e, r4, 0x37));
734.                 //0858:0      not    [op3 0xc10e r4 55 1]
735.                 write3(0xc10e, r4, 0x37, ~read3(0xc10e, r4, 0x37));
736.                 //085d:2      and    [op3 0xf142 r4 35 1], [op3 0xc10e r4 55 1]
737.                 write3(0xf142, r4, 0x23, read3(0xf142, r4, 0x23) & read3(0xc10e, r4, 0x37));
738.                 //0866:1      mov    [op3 0xdbb4 r4 35 1], [op3 0xf142 r4 35 1]
739.                 write3(0xdbb4, r4, 0x23, read3(0xf142, r4, 0x23));
740.                 //086f:0      and    [op3 0xe0b8 r4 18 1], [op3 0xc56e r4 57 1]
741.                 write3(0xe0b8, r4, 0x12, read3(0xe0b8, r4, 0x12) & read3(0xc56e, r4, 0x39));
742.                 //0877:7      mov    [op3 0xc56e r4 57 1], [op3 0xe0b8 r4 18 1]
743.                 write3(0xc56e, r4, 0x39, read3(0xe0b8, r4, 0x12));
744.                 //0880:6      mov    [op3 0xe0b8 r4 18 1], [op3 0xdbb4 r4 35 1]
745.                 write3(0xe0b8, r4, 0x12, read3(0xdbb4, r4, 0x23));
746.                 zz = read3(0xc56e, r4, 0x39) & 0x8000;
747.                 //0889:5      shlf0  [op3 0xc56e r4 57 1], 1
748.                 write3(0xc56e, r4, 0x39, read3(0xc56e, r4, 0x39)<<1);
749.                 if(zz)
750.                 //0890:0      orC0   [op3 0xc56a r4 12 1], 0xbbbc
751.                 {write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc) | 0xbbbc);
752.                 //write3(0xc56e, r4, 0x39, read3(0xc56e, r4, 0x39) & read3(0xc56e, r4, 0x39));
753.                 }
754.             //08a0:4      jNZ0   82a:0 abs
755.             }while(read3(0xc56e, r4, 0x39));
756.
757.             //08a5:5      not    [op3 0xc56a r4 12 1]
758.             write3(0xc56a, r4, 0xc, ~read3(0xc56a, r4, 0xc));
759.             //08aa:7      shlf0  [op3 0xc56a r4 12 1], 1
760.             write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc)<<1);
761.             //08b1:2      mov    [op3 0xec86 r4 37 1], [op3 0xe0b8 r4 18 1]
762.             write3(0xec86, r4, 0x25, read3(0xe0b8, r4, 0x12));
763.             //08ba:1      mov    [op3 0xf282 r4 22 1], 0x2f2c
764.             write3(0xf282, r4, 0x16, 0x2f2c);
765.             //08c1:6      mov    [op3 0xc1c8 r4 61 1], [op3 0xca5c r4 15 1]
766.             write3(0xc1c8, r4, 0x3d, read3(0xca5c, r4, 0xf));
767.             //08ca:5      mov    [op3 0xfde4 r4 35 1], [op3 0xec86 r4 37 1]
768.             write3(0xfde4, r4, 0x23, read3(0xec86, r4, 0x25));
769.             do
770.             {
771.                 //08d3:4      mov    [op3 0xf016 r4 27 1], [op3 0xc1c8 r4 61 1]
772.                 write3(0xf016, r4, 0x1b, read3(0xc1c8, r4, 0x3d));
773.                 //08dc:3      not    [op3 0xf016 r4 27 1]
774.                 write3(0xf016, r4, 0x1b, ~read3(0xf016, r4, 0x1b));
775.                 //08e1:5      and    [op3 0xf016 r4 27 1], [op3 0xfde4 r4 35 1]
776.                 write3(0xf016, r4, 0x1b, read3(0xf016, r4, 0x1b) & read3(0xfde4, r4, 0x23));
777.                 //08ea:4      mov    [op3 0xec30 r4 26 1], [op3 0xc1c8 r4 61 1]
778.                 write3(0xec30, r4, 0x1a, read3(0xc1c8, r4, 0x3d));
779.                 //08f3:3      not    [op3 0xec30 r4 26 1]
780.                 write3(0xec30, r4, 0x1a, ~read3(0xec30, r4, 0x1a));
```

```
781.                //08f8:5      or     [op3 0xec30 r4 26 1], [op3 0xfde4 r4 35 1]
782.                write3(0xec30, r4, 0x1a, read3(0xec30, r4, 0x1a) | read3(0xfde4, r4, 0x23));
783.                //0901:4      not    [op3 0xec30 r4 26 1]
784.                write3(0xec30, r4, 0x1a, ~read3(0xec30, r4, 0x1a));
785.                //0906:6      or     [op3 0xf016 r4 27 1], [op3 0xec30 r4 26 1]
786.                write3(0xf016, r4, 0x1b, read3(0xf016, r4, 0x1b) | read3(0xec30, r4, 0x1a));
787.                //090f:5      mov    [op3 0xc56a r4 12 1], [op3 0xf016 r4 27 1]
788.                write3(0xc56a, r4, 0xc, read3(0xf016, r4, 0x1b));
789.                //0918:4      and    [op3 0xc1c8 r4 61 1], [op3 0xfde4 r4 35 1]
790.                write3(0xc1c8, r4, 0x3d, read3(0xc1c8, r4, 0x3d) & read3(0xfde4, r4, 0x23));
791.                //0921:3      mov    [op3 0xfde4 r4 35 1], [op3 0xc1c8 r4 61 1]
792.                write3(0xfde4, r4, 0x23, read3(0xc1c8, r4, 0x3d));
793.                //092a:2      mov    [op3 0xc1c8 r4 61 1], [op3 0xc56a r4 12 1]
794.                write3(0xc1c8, r4, 0x3d, read3(0xc56a, r4, 0xc));
795.                zz = read3(0xfde4, r4, 0x23) & 0x8000;
796.                //0933:1      shlf1  [op3 0xfde4 r4 35 1], 1
797.                write3(0xfde4, r4, 0x23, read3(0xfde4, r4, 0x23)<<1);
798.                if(zz){
799.                //0939:4      orC1   [op3 0xf282 r4 22 1], 0x87d9
800.                write3(0xf282, r4, 0x16, read3(0xf282, r4, 0x16) | 0x87d9);
801.                //write3(0xfde4, r4, 0x23, read3(0xfde4, r4, 0x23) & read3(0xfde4, r4, 0x23));
802.                }
803.            //094a:0      jNZ1   8d3:4 abs
804.            }while(read3(0xfde4, r4, 0x23) );
805.
806.            //094f:1      not    [op3 0xf282 r4 22 1]
807.            write3(0xf282, r4, 0x16, ~read3(0xf282, r4, 0x16));
808.            //0954:3      shlf1  [op3 0xf282 r4 22 1], 1
809.            write3(0xf282, r4, 0x16, read3(0xf282, r4, 0x16)<<1);
810.            //095a:6      mov    [op3 0xec86 r4 37 1], 0xbe92
811.            write3(0xec86, r4, 0x25, 0xbe92);
812.            //0962:3      mov    [op3 0xc56a r4 12 1], 0x2
813.            write3(0xc56a, r4, 0xc, 0x2);
814.            //096a:0      not    [op3 0xc56a r4 12 1]
815.            write3(0xc56a, r4, 0xc, ~read3(0xc56a, r4, 0xc));
816.            //096f:2      mov    [op3 0xe28c r4 53 1], 0x13b
817.            write3(0xe28c, r4, 0x35, 0x13b);
818.            //0976:7      mov    [op3 0xcf42 r4 16 1], 0x1
819.            write3(0xcf42, r4, 0x10, 0x1);
820.            //097e:4      mov    [op3 0xe402 r4 2 1], [op3 0xc56a r4 12 1]
821.            write3(0xe402, r4, 0x2, read3(0xc56a, r4, 0xc));
822.            do
823.            {
824.                //0987:3      mov    [op3 0xd7dc r4 3 1], [op3 0xcf42 r4 16 1]
825.                write3(0xd7dc, r4, 0x3, read3(0xcf42, r4, 0x10));
826.                //0990:2      not    [op3 0xd7dc r4 3 1]
827.                write3(0xd7dc, r4, 0x3, ~read3(0xd7dc, r4, 0x3));
828.                //0995:4      or     [op3 0xd7dc r4 3 1], [op3 0xe402 r4 2 1]
829.                write3(0xd7dc, r4, 0x3, read3(0xd7dc, r4, 0x3) | read3(0xe402, r4, 0x2));
830.                //099e:3      mov    [op3 0xd98e r4 57 1], [op3 0xcf42 r4 16 1]
831.                write3(0xd98e, r4, 0x39, read3(0xcf42, r4, 0x10));
832.                //09a7:2      mov    [op3 0xddf4 r4 33 1], [op3 0xe402 r4 2 1]
833.                write3(0xddf4, r4, 0x21, read3(0xe402, r4, 0x2));
834.                //09b0:1      not    [op3 0xddf4 r4 33 1]
835.                write3(0xddf4, r4, 0x21, ~read3(0xddf4, r4, 0x21));
836.                //09b5:3      or     [op3 0xd98e r4 57 1], [op3 0xddf4 r4 33 1]
837.                write3(0xd98e, r4, 0x39, read3(0xd98e, r4, 0x39) | read3(0xddf4, r4, 0x21));
838.                //09be:2      and    [op3 0xd7dc r4 3 1], [op3 0xd98e r4 57 1]
839.                write3(0xd7dc, r4, 0x3, read3(0xd7dc, r4, 0x3) & read3(0xd98e, r4, 0x39));
840.                //09c7:1      not    [op3 0xd7dc r4 3 1]
841.                write3(0xd7dc, r4, 0x3, ~read3(0xd7dc, r4, 0x3));
842.                //09cc:3      mov    [op3 0xd6c6 r4 10 1], [op3 0xd7dc r4 3 1]
843.                write3(0xd6c6, r4, 0xa, read3(0xd7dc, r4, 0x3));
844.                //09d5:2      and    [op3 0xcf42 r4 16 1], [op3 0xe402 r4 2 1]
845.                write3(0xcf42, r4, 0x10, read3(0xcf42, r4, 0x10) & read3(0xe402, r4, 0x2));
846.                //09de:1      mov    [op3 0xe402 r4 2 1], [op3 0xcf42 r4 16 1]
847.                write3(0xe402, r4, 0x2, read3(0xcf42, r4, 0x10));
848.                //09e7:0      mov    [op3 0xcf42 r4 16 1], [op3 0xd6c6 r4 10 1]
849.                write3(0xcf42, r4, 0x10, read3(0xd6c6, r4, 0xa));
850.                zz = read3(0xe402, r4, 0x2) & 0x8000;
851.                //09ef:7      shlf0  [op3 0xe402 r4 2 1], 1
852.                write3(0xe402, r4, 0x2, read3(0xe402, r4, 0x2)<<1);
853.                if(zz){
854.                //09f6:2      orC0   [op3 0xe28c r4 53 1], 0xde81
855.                write3(0xe28c, r4, 0x35, read3(0xe28c, r4, 0x35) | 0xde81);
856.                //write3(0xe402, r4, 0x2, read3(0xe402, r4, 0x2) & read3(0xe402, r4, 0x2));
857.                }
858.            //0a06:6      jNZ0   987:3 abs
859.            }while(read3(0xe402, r4, 0x2));
860.
861.            //0a0b:7      not    [op3 0xe28c r4 53 1]
862.            write3(0xe28c, r4, 0x35, ~read3(0xe28c, r4, 0x35));
863.            //0a11:1      shlf0  [op3 0xe28c r4 53 1], 1
864.            write3(0xe28c, r4, 0x35, read3(0xe28c, r4, 0x35)<<1);
865.            //0a17:4      mov    [op3 0xc56a r4 12 1], [op3 0xcf42 r4 16 1]
866.            write3(0xc56a, r4, 0xc, read3(0xcf42, r4, 0x10));
867.            //0a20:3      mov    [op3 0xeda0 r4 34 1], 0x7500
868.            write3(0xeda0, r4, 0x22, 0x7500);
```

```
            //0a28:0      mov    [op3 0xc9c8 r4 48 1], r8
            write3(0xc9c8, r4, 0x30, r8);
            //0a2e:1      mov    [op3 0xc4cc r4 6 1], [op3 0xc56a r4 12 1]
            write3(0xc4cc, r4, 0x6, read3(0xc56a, r4, 0xc));
            do
            {
                //0a37:0      mov    [op3 0xd3b0 r4 24 1], [op3 0xc9c8 r4 48 1]
                write3(0xd3b0, r4, 0x18, read3(0xc9c8, r4, 0x30));
                //0a3f:7      mov    [op3 0xc372 r4 16 1], [op3 0xc4cc r4 6 1]
                write3(0xc372, r4, 0x10, read3(0xc4cc, r4, 0x6));
                //0a48:6      not    [op3 0xc372 r4 16 1]
                write3(0xc372, r4, 0x10, ~read3(0xc372, r4, 0x10));
                //0a4e:0      or     [op3 0xd3b0 r4 24 1], [op3 0xc372 r4 16 1]
                write3(0xd3b0, r4, 0x18, read3(0xd3b0, r4, 0x18) | read3(0xc372, r4, 0x10));
                //0a56:7      mov    [op3 0xd17e r4 59 1], [op3 0xc9c8 r4 48 1]
                write3(0xd17e, r4, 0x3b, read3(0xc9c8, r4, 0x30));
                //0a5f:6      not    [op3 0xd17e r4 59 1]
                write3(0xd17e, r4, 0x3b, ~read3(0xd17e, r4, 0x3b));
                //0a65:0      or     [op3 0xd17e r4 59 1], [op3 0xc4cc r4 6 1]
                write3(0xd17e, r4, 0x3b, read3(0xd17e, r4, 0x3b) | read3(0xc4cc, r4, 0x6));
                //0a6d:7      and    [op3 0xd3b0 r4 24 1], [op3 0xd17e r4 59 1]
                write3(0xd3b0, r4, 0x18, read3(0xd3b0, r4, 0x18) & read3(0xd17e, r4, 0x3b));
                //0a76:6      not    [op3 0xd3b0 r4 24 1]
                write3(0xd3b0, r4, 0x18, ~read3(0xd3b0, r4, 0x18));
                //0a7c:0      mov    [op3 0xe7a6 r4 9 1], [op3 0xd3b0 r4 24 1]
                write3(0xe7a6, r4, 0x9, read3(0xd3b0, r4, 0x18));
                //0a84:7      and    [op3 0xc9c8 r4 48 1], [op3 0xc4cc r4 6 1]
                write3(0xc9c8, r4, 0x30, read3(0xc9c8, r4, 0x30) & read3(0xc4cc, r4, 0x6));
                //0a8d:6      mov    [op3 0xc4cc r4 6 1], [op3 0xc9c8 r4 48 1]
                write3(0xc4cc, r4, 0x6, read3(0xc9c8, r4, 0x30));
                //0a96:5      mov    [op3 0xc9c8 r4 48 1], [op3 0xe7a6 r4 9 1]
                write3(0xc9c8, r4, 0x30, read3(0xe7a6, r4, 0x9));
                zz = read3(0xc4cc, r4, 0x6) & 0x8000;
                //0a9f:4      shlf1  [op3 0xc4cc r4 6 1], 1
                write3(0xc4cc, r4, 0x6, read3(0xc4cc, r4, 0x6)<<1);
                if(zz)
                {
                //0aa5:7      orC1   [op3 0xeda0 r4 34 1], 0x9ad9
                write3(0xeda0, r4, 0x22, read3(0xeda0, r4, 0x22) | 0x9ad9);
                //write3(0xc4cc, r4, 0x6, read3(0xc4cc, r4, 0x6) & read3(0xc4cc, r4, 0x6));
                }
            //0ab6:3      jNZ1   a37:0 abs
            }while(read3(0xc4cc, r4, 0x6));

            //0abb:4      not    [op3 0xeda0 r4 34 1]
            write3(0xeda0, r4, 0x22, ~read3(0xeda0, r4, 0x22));
            //0ac0:6      shlf1  [op3 0xeda0 r4 34 1], 1
            write3(0xeda0, r4, 0x22, read3(0xeda0, r4, 0x22)<<1);
            //0ac7:1      mov    r8, [op3 0xc9c8 r4 48 1]
            r8 = read3(0xc9c8, r4, 0x30);
            //0acd:2      mov    [op3 0xd19c r4 40 1], 1[0x9fc0 + r8]
            write3(0xd19c, r4, 0x28, *((uint16_t*)&ram[0x9fc0 + r8]));
            //0ad5:5      mov    [op3 0xfeda r4 53 1], [op3 0xd19c r4 40 1]
            write3(0xfeda, r4, 0x35, read3(0xd19c, r4, 0x28));
            //0ade:4      mov    [op3 0xccf6 r4 32 1], [op3 0xec86 r4 37 1]
            write3(0xccf6, r4, 0x20, read3(0xec86, r4, 0x25));
            //0ae7:3      not    [op3 0xccf6 r4 32 1]
            write3(0xccf6, r4, 0x20, ~read3(0xccf6, r4, 0x20));
            //0aec:5      mov    [op3 0xf346 r4 21 1], 0x1d75
            write3(0xf346, r4, 0x15, 0x1d75);
            //0af4:2      mov    [op3 0xc966 r4 39 1], 0x1
            write3(0xc966, r4, 0x27, 0x1);
            //0afb:7      mov    [op3 0xd092 r4 23 1], [op3 0xccf6 r4 32 1]
            write3(0xd092, r4, 0x17, read3(0xccf6, r4, 0x20));
            do
            {
                //0b04:6      mov    [op3 0xd80c r4 44 1], [op3 0xc966 r4 39 1]
                write3(0xd80c, r4, 0x2c, read3(0xc966, r4, 0x27));
                //0b0d:5      and    [op3 0xd80c r4 44 1], [op3 0xd092 r4 23 1]
                write3(0xd80c, r4, 0x2c, read3(0xd80c, r4, 0x2c) & read3(0xd092, r4, 0x17));
                //0b16:4      not    [op3 0xd80c r4 44 1]
                write3(0xd80c, r4, 0x2c, ~read3(0xd80c, r4, 0x2c));
                //0b1b:6      mov    [op3 0xc2b6 r4 33 1], [op3 0xc966 r4 39 1]
                write3(0xc2b6, r4, 0x21, read3(0xc966, r4, 0x27));
                //0b24:5      or     [op3 0xc2b6 r4 33 1], [op3 0xd092 r4 23 1]
                write3(0xc2b6, r4, 0x21, read3(0xc2b6, r4, 0x21) | read3(0xd092, r4, 0x17));
                //0b2d:4      not    [op3 0xc2b6 r4 33 1]
                write3(0xc2b6, r4, 0x21, ~read3(0xc2b6, r4, 0x21));
                //0b32:6      not    [op3 0xc2b6 r4 33 1]
                write3(0xc2b6, r4, 0x21, ~read3(0xc2b6, r4, 0x21));
                //0b38:0      and    [op3 0xd80c r4 44 1], [op3 0xc2b6 r4 33 1]
                write3(0xd80c, r4, 0x2c, read3(0xd80c, r4, 0x2c) & read3(0xc2b6, r4, 0x21));
                //0b40:7      mov    [op3 0xc56a r4 12 1], [op3 0xd80c r4 44 1]
                write3(0xc56a, r4, 0xc, read3(0xd80c, r4, 0x2c));
                //0b49:6      and    [op3 0xc966 r4 39 1], [op3 0xd092 r4 23 1]
                write3(0xc966, r4, 0x27, read3(0xc966, r4, 0x27) & read3(0xd092, r4, 0x17));
                //0b52:5      mov    [op3 0xd092 r4 23 1], [op3 0xc966 r4 39 1]
                write3(0xd092, r4, 0x17, read3(0xc966, r4, 0x27));
```

```
957.                //0b5b:4      mov    [op3 0xc966 r4 39 1], [op3 0xc56a r4 12 1]
958.                write3(0xc966, r4, 0x27, read3(0xc56a, r4, 0xc));
959.                zz = read3(0xd092, r4, 0x17) & 0x8000;
960.                //0b64:3      shlf1  [op3 0xd092 r4 23 1], 1
961.                write3(0xd092, r4, 0x17, read3(0xd092, r4, 0x17)<<1);
962.                if(zz)
963.                //0b6a:6      orC1   [op3 0xf346 r4 21 1], 0xa04f
964.                {write3(0xf346, r4, 0x15, read3(0xf346, r4, 0x15) | 0xa04f);
965.                //write3(0xd092, r4, 0x17, read3(0xd092, r4, 0x17) & read3(0xd092, r4, 0x17));
966.                }
967.            //0b7b:2      jNZ1   b04:6 abs
968.            }while(read3(0xd092, r4, 0x17));
969.            //0b80:3      not    [op3 0xf346 r4 21 1]
970.            write3(0xf346, r4, 0x15, ~read3(0xf346, r4, 0x15));
971.            //0b85:5      shlf1  [op3 0xf346 r4 21 1], 1
972.            write3(0xf346, r4, 0x15, read3(0xf346, r4, 0x15)<<1);
973.            //0b8c:0      mov    [op3 0xccf6 r4 32 1], [op3 0xc966 r4 39 1]
974.            write3(0xccf6, r4, 0x20, read3(0xc966, r4, 0x27));
975.            //0b94:7      mov    [op3 0xc46a r4 24 1], 0xda1
976.            write3(0xc46a, r4, 0x18, 0xda1);
977.            //0b9c:4      mov    [op3 0xfb4e r4 53 1], [op3 0xfeda r4 53 1]
978.            write3(0xfb4e, r4, 0x35, read3(0xfeda, r4, 0x35));
979.            //0ba5:3      mov    [op3 0xd072 r4 59 1], [op3 0xccf6 r4 32 1]
980.            write3(0xd072, r4, 0x3b, read3(0xccf6, r4, 0x20));
981.            do
982.            {
983.                //0bae:2      mov    [op3 0xf5de r4 43 1], [op3 0xfb4e r4 53 1]
984.                write3(0xf5de, r4, 0x2b, read3(0xfb4e, r4, 0x35));
985.                //0bb7:1      mov    [op3 0xe000 r4 38 1], [op3 0xd072 r4 59 1]
986.                write3(0xe000, r4, 0x26, read3(0xd072, r4, 0x3b));
987.                //0bc0:0      not    [op3 0xe000 r4 38 1]
988.                write3(0xe000, r4, 0x26, ~read3(0xe000, r4, 0x26));
989.                //0bc5:2      and    [op3 0xf5de r4 43 1], [op3 0xe000 r4 38 1]
990.                write3(0xf5de, r4, 0x2b, read3(0xf5de, r4, 0x2b) & read3(0xe000, r4, 0x26));
991.                //0bce:1      mov    [op3 0xfd18 r4 54 1], [op3 0xfb4e r4 53 1]
992.                write3(0xfd18, r4, 0x36, read3(0xfb4e, r4, 0x35));
993.                //0bd7:0      mov    [op3 0xc698 r4 29 1], [op3 0xd072 r4 59 1]
994.                write3(0xc698, r4, 0x1d, read3(0xd072, r4, 0x3b));
995.                //0bdf:7      not    [op3 0xc698 r4 29 1]
996.                write3(0xc698, r4, 0x1d, ~read3(0xc698, r4, 0x1d));
997.                //0be5:1      or     [op3 0xfd18 r4 54 1], [op3 0xc698 r4 29 1]
998.                write3(0xfd18, r4, 0x36, read3(0xfd18, r4, 0x36) | read3(0xc698, r4, 0x1d));
999.                //0bee:0      not    [op3 0xfd18 r4 54 1]
1000.                   write3(0xfd18, r4, 0x36, ~read3(0xfd18, r4, 0x36));
1001.                   //0bf3:2      or     [op3 0xf5de r4 43 1], [op3 0xfd18 r4 54 1]
1002.                   write3(0xf5de, r4, 0x2b, read3(0xf5de, r4, 0x2b) | read3(0xfd18, r4, 0x36));
1003.                   //0bfc:1      mov    [op3 0xc56a r4 12 1], [op3 0xf5de r4 43 1]
1004.                   write3(0xc56a, r4, 0xc, read3(0xf5de, r4, 0x2b));
1005.                   //0c05:0      and    [op3 0xfb4e r4 53 1], [op3 0xd072 r4 59 1]
1006.                   write3(0xfb4e, r4, 0x35, read3(0xfb4e, r4, 0x35) & read3(0xd072, r4, 0x3b));
1007.                   //0c0d:7      mov    [op3 0xd072 r4 59 1], [op3 0xfb4e r4 53 1]
1008.                   write3(0xd072, r4, 0x3b, read3(0xfb4e, r4, 0x35));
1009.                   //0c16:6      mov    [op3 0xfb4e r4 53 1], [op3 0xc56a r4 12 1]
1010.                   write3(0xfb4e, r4, 0x35, read3(0xc56a, r4, 0xc));
1011.                   zz = read3(0xd072, r4, 0x3b) & 0x8000;
1012.                   //0c1f:5      shlf1  [op3 0xd072 r4 59 1], 1
1013.                   write3(0xd072, r4, 0x3b, read3(0xd072, r4, 0x3b)<<1);
1014.                   if(zz)
1015.                   //0c26:0      orC1   [op3 0xc46a r4 24 1], 0xc36e
1016.                   {write3(0xc46a, r4, 0x18, read3(0xc46a, r4, 0x18) | 0xc36e);
1017.                   //write3(0xd072, r4, 0x3b, read3(0xd072, r4, 0x3b) & read3(0xd072, r4, 0x3b));
1018.                   }
1019.               //0c36:4      jNZ1   bae:2 abs
1020.               }while(read3(0xd072, r4, 0x3b));
1021.               //0c3b:5      not    [op3 0xc46a r4 24 1]
1022.               write3(0xc46a, r4, 0x18, ~read3(0xc46a, r4, 0x18));
1023.               //0c40:7      shlf1  [op3 0xc46a r4 24 1], 1
1024.               write3(0xc46a, r4, 0x18, read3(0xc46a, r4, 0x18)<<1);
1025.               //0c47:2      mov    [op3 0xfeda r4 53 1], [op3 0xfb4e r4 53 1]
1026.               write3(0xfeda, r4, 0x35, read3(0xfb4e, r4, 0x35));
1027.       //write3(0xfeda, r4, 0x35, read3(0xfeda, r4, 0x35) & read3(0xfeda, r4, 0x35));
1028.
1029.               if(read3(0xfeda, r4, 0x35))
1030.               {
1031.                   //0c59:0      movNZ1 1[0x9fc0 + r8], 0xffff
1032.                   *((uint16_t*)&ram[0x9fc0 + r8]) = 0xffff;
1033.                   return 0;
1034.       //var_9fc0 + r8 = 0xffff;
1035.
1036.               }
1037.            }
1038.            //0c60:1      mov    [op3 0xd7ac r4 33 1], [op3 0xdf44 r4 55 1]
1039.            write3(0xd7ac, r4, 0x21, read3(0xdf44, r4, 0x37));
1040.            //0c69:0      mov    [op3 0xd5e8 r4 13 1], 0x40
1041.            write3(0xd5e8, r4, 0xd, 0x40);
1042.            //0c70:5      not    [op3 0xd5e8 r4 13 1]
1043.            write3(0xd5e8, r4, 0xd, ~read3(0xd5e8, r4, 0xd));
1044.            //0c75:7      mov    [op3 0xc636 r4 35 1], 0x5020
```

```
1045.                 write3(0xc636, r4, 0x23, 0x5020);
1046.                 //0c7d:4      mov    [op3 0xfe82 r4 14 1], 0x1
1047.                 write3(0xfe82, r4, 0xe, 0x1);
1048.                 //0c85:1      mov    [op3 0xc958 r4 48 1], [op3 0xd5e8 r4 13 1]
1049.                 write3(0xc958, r4, 0x30, read3(0xd5e8, r4, 0xd));
1050.                 do
1051.                 {
1052.                     //0c8e:0      mov    [op3 0xc642 r4 7 1], [op3 0xfe82 r4 14 1]
1053.                     write3(0xc642, r4, 0x7, read3(0xfe82, r4, 0xe));
1054.                     //0c96:7      mov    [op3 0xf9ea r4 42 1], [op3 0xc958 r4 48 1]
1055.                     write3(0xf9ea, r4, 0x2a, read3(0xc958, r4, 0x30));
1056.                     //0c9f:6      not    [op3 0xf9ea r4 42 1]
1057.                     write3(0xf9ea, r4, 0x2a, ~read3(0xf9ea, r4, 0x2a));
1058.                     //0ca5:0      and    [op3 0xc642 r4 7 1], [op3 0xf9ea r4 42 1]
1059.                     write3(0xc642, r4, 0x7, read3(0xc642, r4, 0x7) & read3(0xf9ea, r4, 0x2a));
1060.                     //0cad:7      mov    [op3 0xf9ea r4 42 1], [op3 0xc642 r4 7 1]
1061.                     write3(0xf9ea, r4, 0x2a, read3(0xc642, r4, 0x7));
1062.                     //0cb6:6      mov    [op3 0xc56a r4 12 1], [op3 0xfe82 r4 14 1]
1063.                     write3(0xc56a, r4, 0xc, read3(0xfe82, r4, 0xe));
1064.                     //0cbf:5      mov    [op3 0xfac2 r4 42 1], [op3 0xc958 r4 48 1]
1065.                     write3(0xfac2, r4, 0x2a, read3(0xc958, r4, 0x30));
1066.                     //0cc8:4      not    [op3 0xfac2 r4 42 1]
1067.                     write3(0xfac2, r4, 0x2a, ~read3(0xfac2, r4, 0x2a));
1068.                     //0ccd:6      or     [op3 0xc56a r4 12 1], [op3 0xfac2 r4 42 1]
1069.                     write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc) | read3(0xfac2, r4, 0x2a));
1070.                     //0cd6:5      not    [op3 0xc56a r4 12 1]
1071.                     write3(0xc56a, r4, 0xc, ~read3(0xc56a, r4, 0xc));
1072.                     //0cdb:7      or     [op3 0xf9ea r4 42 1], [op3 0xc56a r4 12 1]
1073.                     write3(0xf9ea, r4, 0x2a, read3(0xf9ea, r4, 0x2a) | read3(0xc56a, r4, 0xc));
1074.                     //0ce4:6      mov    [op3 0xc122 r4 16 1], [op3 0xf9ea r4 42 1]
1075.                     write3(0xc122, r4, 0x10, read3(0xf9ea, r4, 0x2a));
1076.                     //0ced:5      and    [op3 0xfe82 r4 14 1], [op3 0xc958 r4 48 1]
1077.                     write3(0xfe82, r4, 0xe, read3(0xfe82, r4, 0xe) & read3(0xc958, r4, 0x30));
1078.                     //0cf6:4      mov    [op3 0xc958 r4 48 1], [op3 0xfe82 r4 14 1]
1079.                     write3(0xc958, r4, 0x30, read3(0xfe82, r4, 0xe));
1080.                     //0cff:3      mov    [op3 0xfe82 r4 14 1], [op3 0xc122 r4 16 1]
1081.                     write3(0xfe82, r4, 0xe, read3(0xc122, r4, 0x10));
1082.                     zz = read3(0xc958, r4, 0x30) & 0x8000;
1083.                     //0d08:2      shlf1  [op3 0xc958 r4 48 1], 1
1084.                     write3(0xc958, r4, 0x30, read3(0xc958, r4, 0x30)<<1);
1085.                     if(zz){
1086.                     //0d0e:5      orC1   [op3 0xc636 r4 35 1], 0x9bce
1087.                     write3(0xc636, r4, 0x23, read3(0xc636, r4, 0x23) | 0x9bce);
1088.                     }
1089.         //write3(0xc958, r4, 0x30, read3(0xc958, r4, 0x30) & read3(0xc958, r4, 0x30));
1090.
1091.                 //0d1f:1      jNZ1   c8e:0 abs
1092.                 }while(read3(0xc958, r4, 0x30));
1093.                 //0d24:2      not    [op3 0xc636 r4 35 1]
1094.                 write3(0xc636, r4, 0x23, ~read3(0xc636, r4, 0x23));
1095.                 //0d29:4      shlf1  [op3 0xc636 r4 35 1], 1
1096.                 write3(0xc636, r4, 0x23, read3(0xc636, r4, 0x23)<<1);
1097.                 //0d2f:7      mov    [op3 0xd5e8 r4 13 1], [op3 0xfe82 r4 14 1]
1098.                 write3(0xd5e8, r4, 0xd, read3(0xfe82, r4, 0xe));
1099.                 //0d38:6      mov    [op3 0xc56a r4 12 1], 0x5075
1100.                 write3(0xc56a, r4, 0xc, 0x5075);
1101.                 //0d40:3      mov    [op3 0xfee4 r4 1 1], [op3 0xd7ac r4 33 1]
1102.                 write3(0xfee4, r4, 0x1, read3(0xd7ac, r4, 0x21));
1103.                 //0d49:2      mov    [op3 0xc402 r4 61 1], [op3 0xd5e8 r4 13 1]
1104.                 write3(0xc402, r4, 0x3d, read3(0xd5e8, r4, 0xd));
1105.                 do
1106.                 {
1107.                     //0d52:1      mov    [op3 0xf8b0 r4 1 1], [op3 0xfee4 r4 1 1]
1108.                     write3(0xf8b0, r4, 0x1, read3(0xfee4, r4, 0x1));
1109.                     //0d5b:0      or     [op3 0xf8b0 r4 1 1], [op3 0xc402 r4 61 1]
1110.                     write3(0xf8b0, r4, 0x1, read3(0xf8b0, r4, 0x1) | read3(0xc402, r4, 0x3d));
1111.                     //0d63:7      mov    [op3 0xfe56 r4 3 1], [op3 0xfee4 r4 1 1]
1112.                     write3(0xfe56, r4, 0x3, read3(0xfee4, r4, 0x1));
1113.                     //0d6c:6      and    [op3 0xfe56 r4 3 1], [op3 0xc402 r4 61 1]
1114.                     write3(0xfe56, r4, 0x3, read3(0xfe56, r4, 0x3) & read3(0xc402, r4, 0x3d));
1115.                     //0d75:5      not    [op3 0xfe56 r4 3 1]
1116.                     write3(0xfe56, r4, 0x3, ~read3(0xfe56, r4, 0x3));
1117.                     //0d7a:7      and    [op3 0xf8b0 r4 1 1], [op3 0xfe56 r4 3 1]
1118.                     write3(0xf8b0, r4, 0x1, read3(0xf8b0, r4, 0x1) & read3(0xfe56, r4, 0x3));
1119.                     //0d83:6      mov    [op3 0xec06 r4 55 1], [op3 0xf8b0 r4 1 1]
1120.                     write3(0xec06, r4, 0x37, read3(0xf8b0, r4, 0x1));
1121.                     //0d8c:5      and    [op3 0xfee4 r4 1 1], [op3 0xc402 r4 61 1]
1122.                     write3(0xfee4, r4, 0x1, read3(0xfee4, r4, 0x1) & read3(0xc402, r4, 0x3d));
1123.                     //0d95:4      mov    [op3 0xc402 r4 61 1], [op3 0xfee4 r4 1 1]
1124.                     write3(0xc402, r4, 0x3d, read3(0xfee4, r4, 0x1));
1125.                     //0d9e:3      mov    [op3 0xfee4 r4 1 1], [op3 0xec06 r4 55 1]
1126.                     write3(0xfee4, r4, 0x1, read3(0xec06, r4, 0x37));
1127.                     zz = read3(0xc402, r4, 0x3d) & 0x8000;
1128.                     //0da7:2      shlf0  [op3 0xc402 r4 61 1], 1
1129.                     write3(0xc402, r4, 0x3d, read3(0xc402, r4, 0x3d)<<1);
1130.                     if(zz)
1131.                     //0dad:5      orC0   [op3 0xc56a r4 12 1], 0xed90
1132.                     {write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc) | 0xed90);
```

```
                    //write3(0xc402, r4, 0x3d, read3(0xc402, r4, 0x3d) & read3(0xc402, r4, 0x3d));
                    }
                //0dbe:1      jNZ0   d52:1 abs
                }while(read3(0xc402, r4, 0x3d));
                //0dc3:2      not    [op3 0xc56a r4 12 1]
                write3(0xc56a, r4, 0xc, ~read3(0xc56a, r4, 0xc));
                zz = read3(0xc56a, r4, 0xc) & 0x8000;
                //0dc8:4      shlf0  [op3 0xc56a r4 12 1], 1
                write3(0xc56a, r4, 0xc, read3(0xc56a, r4, 0xc)<<1);
                //0dce:7      mov    [op3 0xd7ac r4 33 1], [op3 0xfee4 r4 1 1]
                write3(0xd7ac, r4, 0x21, read3(0xfee4, r4, 0x1));
                //0dd7:6      andf0  [op3 0xd7ac r4 33 1], [op3 0xd7ac r4 33 1]
                write3(0xd7ac, r4, 0x21, read3(0xd7ac, r4, 0x21) & read3(0xd7ac, r4, 0x21));
            }while(zz);

        //0de0:5      jC0    1f3:3 abs

            //0de5:6      mov    [0x8000], 0xdead
            *((uint16_t*)&ram[0x8000]) = 0xdead;
            //0dec:3      mov    [op3 0xd458 r4 31 1], r9
            write3(0xd458, r4, 0x1f, r9);
            //0df2:4      and    [op3 0xd458 r4 31 1], r11
            write3(0xd458, r4, 0x1f, read3(0xd458, r4, 0x1f) & r11);
            //0df8:5      mov    [0x8002], 0xbeef
            *((uint16_t*)&ram[0x8002]) = 0xbeef;

            char blobfilename[100];
            sprintf(blobfilename, "blob_%x_%x.bin", var_a000, var_a002);
            printf("Creating file %s\n", blobfilename);
            int fd = open(blobfilename, O_CREAT | O_WRONLY, 0777);
            write(fd, &ram[0xa010], 256);
            close(fd);
            return 1;
        }

        int main(int argc, char** argv)
        {
            uint16_t i;

            for(i=0xffff; i > 0 ; i--)
            {
                if(check_key2(0x94e3, ((uint16_t*)&i)[0]))
                {
                    printf("Key? 0x%x 0x%x\n", 0x94e3, i);
                    break;
                }
            }
            return 0;
        }
```

LAYER3.C

```
//gcc -Wall -O3 layer3.c -o layer3
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

unsigned char blah[32] = {
    0xBA, 0x64, 0x13, 0x54, 0x97, 0x70, 0x73, 0x0E, 0xFA, 0x62, 0x61, 0x19,
    0x56, 0x57, 0x38, 0x31, 0xC3, 0xC9, 0x52, 0x36, 0x5B, 0x3D, 0x6E, 0xAF,
    0x06, 0x56, 0x49, 0xB7, 0x0F, 0x4D, 0x54, 0xAC
};

uint8_t ram[0x10000]={0};

static unsigned int rol16(unsigned int i, int c) {
    return ((i << c) & 65535) | ((unsigned int)(i & 65535) >> (16 - c));
}
static unsigned int ror16(unsigned int i, int c) {
    return ((unsigned int)(i & 65535) >> c) | ((i << (16 - c)) & 65535);
}

uint16_t write3(uint16_t imm16, uint16_t reg, uint16_t imm6, uint16_t val)
{
    uint16_t r5 = ((((imm6 << 6) + imm6) << 4) + imm6) ^ 0x464d;
    uint16_t r2 = (imm16 ^ 0x6c38);
    uint16_t r6 = 0;
    uint16_t r7 = 0;
    imm16 += reg;
```

```
    reg += 2;
    do
    {
        r7 = r6;
        r5 = rol16(r5,1);
        r2 = ror16(r2,2) + r5;
        r5 += 2;
        r6 = r2 ^ r5;
        r6 = ((r6 >> 8 ) & 0xff) ^ (r6 & 0xff);
        reg--;
    }while(reg);

    r6 <<= 8;
    r6 |= r7;

    //printf("write at %x = %x\n", imm16, val ^ r6);
    *((uint16_t*)&ram[imm16]) = val ^ r6;
    return 0;
}

uint16_t read3(uint16_t imm16, uint16_t reg, uint16_t imm6)
{
    uint16_t r5 = ((((imm6 << 6) + imm6) << 4) + imm6) ^ 0x464d;
    uint16_t r2 = (imm16 ^ 0x6c38);
    uint16_t r6 = 0;
    uint16_t r7 = 0;
    imm16 += reg;
    reg += 2;
    do
    {
        r7 = r6;
        r5 = rol16(r5,1);
        r2 = ror16(r2,2) + r5;
        r5 += 2;
        r6 = r2 ^ r5;
        r6 = ((r6 >> 8 ) & 0xff) ^ (r6 & 0xff);
        reg--;
    }while(reg);

    r6 <<= 8;
    r6 |= r7;

    uint16_t res = *((uint16_t*)&ram[imm16]) ^ r6;
    //printf("read %x = %x\n", imm16, res);
    return res;
}
uint16_t r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13,r14;

uint16_t zz;

uint16_t check_key3(uint16_t var_a000, uint16_t var_a002)
{
    memcpy(&ram[0xA000], &var_a000, 2);
    memcpy(&ram[0xA002], &var_a002, 2);
    memcpy(&ram[0xA010], blah, 32);


    //0000:0      mov    r13, r7
    r13 = r7;
    //0003:3      and    r9, 0x0
    r9 &= 0x0;
    //0008:2      mov    [op3 0xdb2a r9 59 1], [0xa000]
    write3(0xdb2a, r9, 0x3b, *((uint16_t*)&ram[0xa000]));
    //0010:1      mov    [op3 0xd73c r9 40 1], r2
    write3(0xd73c, r9, 0x28, r2);
    //0016:2      or     [op3 0xd73c r9 40 1], r0
    write3(0xd73c, r9, 0x28, read3(0xd73c, r9, 0x28) | r0);
    //001c:3      mov    r0, [0xa002]
    r0 = *((uint16_t*)&ram[0xa002]);
    //0021:4      mov    [op3 0xd9a6 r9 24 1], 0x6e63
    write3(0xd9a6, r9, 0x18, 0x6e63);
    //0029:1      shr    [op3 0xd9a6 r9 24 1], 1
    write3(0xd9a6, r9, 0x18, read3(0xd9a6, r9, 0x18)>>1);
    //002f:4      mov    r12, [op3 0xdb2a r9 59 1]
    r12 = read3(0xdb2a, r9, 0x3b);
    //0035:5      and    r12, 0xff
    r12 &= 0xff;
    //003a:4      mov    [op3 0xf9c4 r9 41 1], r12
    write3(0xf9c4, r9, 0x29, r12);
    //0040:5      mov    [op3 0xdf0c r9 7 1], r0
    write3(0xdf0c, r9, 0x7, r0);
    //0046:6      and    [op3 0xdf0c r9 7 1], 0xff
    write3(0xdf0c, r9, 0x7, read3(0xdf0c, r9, 0x7) & 0xff);
    //004e:3      mov    [op3 0xef0c r9 19 1], r4
    write3(0xef0c, r9, 0x13, r4);
    //0054:4      or     [op3 0xef0c r9 19 1], r7
    write3(0xef0c, r9, 0x13, read3(0xef0c, r9, 0x13) | r7);
    //005a:5      mov    [op3 0xc910 r9 12 1], [op3 0xf9c4 r9 41 1]
```

```
    write3(0xc910, r9, 0xc, read3(0xf9c4, r9, 0x29));
    //0063:4      and    [op3 0xc910 r9 12 1], [op3 0xdf0c r9 7 1]
    write3(0xc910, r9, 0xc, read3(0xc910, r9, 0xc) & read3(0xdf0c, r9, 0x7));
    //006c:3      not    [op3 0xc910 r9 12 1]
    write3(0xc910, r9, 0xc, ~read3(0xc910, r9, 0xc));
    //0071:5      mov    [op3 0xfdee r9 53 1], [op3 0xf9c4 r9 41 1]
    write3(0xfdee, r9, 0x35, read3(0xf9c4, r9, 0x29));
    //007a:4      or     [op3 0xfdee r9 53 1], [op3 0xdf0c r9 7 1]
    write3(0xfdee, r9, 0x35, read3(0xfdee, r9, 0x35) | read3(0xdf0c, r9, 0x7));
    //0083:3      and    [op3 0xc910 r9 12 1], [op3 0xfdee r9 53 1]
    write3(0xc910, r9, 0xc, read3(0xc910, r9, 0xc) & read3(0xfdee, r9, 0x35));
    //008c:2      mov    r13, [op3 0xc910 r9 12 1]
    r13 = read3(0xc910, r9, 0xc);
    //0092:3      mov    [op3 0xfb78 r9 46 1], [op3 0xdb2a r9 59 1]
    write3(0xfb78, r9, 0x2e, read3(0xdb2a, r9, 0x3b));
    //009b:2      shr    [op3 0xfb78 r9 46 1], 8
    write3(0xfb78, r9, 0x2e, read3(0xfb78, r9, 0x2e)>>8);
    //00a1:5      mov    [op3 0xf9c4 r9 41 1], [op3 0xfb78 r9 46 1]
    write3(0xf9c4, r9, 0x29, read3(0xfb78, r9, 0x2e));
    //00aa:4      mov    [op3 0xfb78 r9 46 1], r2
    write3(0xfb78, r9, 0x2e, r2);
    //00b0:5      mov    [op3 0xe462 r9 35 1], r0
    write3(0xe462, r9, 0x23, r0);
    //00b6:6      shr    [op3 0xe462 r9 35 1], 8
    write3(0xe462, r9, 0x23, read3(0xe462, r9, 0x23)>>8);
    //00bd:1      mov    [op3 0xdf0c r9 7 1], [op3 0xe462 r9 35 1]
    write3(0xdf0c, r9, 0x7, read3(0xe462, r9, 0x23));
    //00c6:0      mov    [op3 0xdfc8 r9 53 1], [op3 0xf9c4 r9 41 1]
    write3(0xdfc8, r9, 0x35, read3(0xf9c4, r9, 0x29));
    //00ce:7      not    [op3 0xdfc8 r9 53 1]
    write3(0xdfc8, r9, 0x35, ~read3(0xdfc8, r9, 0x35));
    //00d4:1      and    [op3 0xdfc8 r9 53 1], [op3 0xdf0c r9 7 1]
    write3(0xdfc8, r9, 0x35, read3(0xdfc8, r9, 0x35) & read3(0xdf0c, r9, 0x7));
    //00dd:0      mov    [op3 0xea26 r9 49 1], [op3 0xf9c4 r9 41 1]
    write3(0xea26, r9, 0x31, read3(0xf9c4, r9, 0x29));
    //00e5:7      not    [op3 0xea26 r9 49 1]
    write3(0xea26, r9, 0x31, ~read3(0xea26, r9, 0x31));
    //00eb:1      or     [op3 0xea26 r9 49 1], [op3 0xdf0c r9 7 1]
    write3(0xea26, r9, 0x31, read3(0xea26, r9, 0x31) | read3(0xdf0c, r9, 0x7));
    //00f4:0      not    [op3 0xea26 r9 49 1]
    write3(0xea26, r9, 0x31, ~read3(0xea26, r9, 0x31));
    //00f9:2      or     [op3 0xdfc8 r9 53 1], [op3 0xea26 r9 49 1]
    write3(0xdfc8, r9, 0x35, read3(0xdfc8, r9, 0x35) | read3(0xea26, r9, 0x31));
    //0102:1      mov    r7, [op3 0xdfc8 r9 53 1]
    r7 = read3(0xdfc8, r9, 0x35);
    //0108:2      mov    [op3 0xf390 r9 6 1], r8
    write3(0xf390, r9, 0x6, r8);
    //010e:3      mov    [op3 0xf740 r9 49 1], r7
    write3(0xf740, r9, 0x31, r7);
    //0114:4      shl    [op3 0xf740 r9 49 1], 8
    write3(0xf740, r9, 0x31, read3(0xf740, r9, 0x31)<<8);
    //011a:7      mov    [op3 0xdb2a r9 59 1], [op3 0xf740 r9 49 1]
    write3(0xdb2a, r9, 0x3b, read3(0xf740, r9, 0x31));
    //0123:6      mov    [op3 0xf740 r9 49 1], r11
    write3(0xf740, r9, 0x31, r11);
    //0129:7      mov    [op3 0xd944 r9 13 1], 0x4cf0
    write3(0xd944, r9, 0xd, 0x4cf0);
    //0131:4      mov    [op3 0xf49c r9 2 1], r7
    write3(0xf49c, r9, 0x2, r7);
    //0137:5      mov    [op3 0xe216 r9 12 1], r13
    write3(0xe216, r9, 0xc, r13);
    do
    {
        //013d:6      mov    r12, [op3 0xf49c r9 2 1]
        r12 = read3(0xf49c, r9, 0x2);
        //0143:7      mov    [op3 0xe5a0 r9 43 1], [op3 0xe216 r9 12 1]
        write3(0xe5a0, r9, 0x2b, read3(0xe216, r9, 0xc));
        //014c:6      not    [op3 0xe5a0 r9 43 1]
        write3(0xe5a0, r9, 0x2b, ~read3(0xe5a0, r9, 0x2b));
        //0152:0      and    r12, [op3 0xe5a0 r9 43 1]
        r12 &= read3(0xe5a0, r9, 0x2b);
        //0158:1      mov    [op3 0xe5a0 r9 43 1], r12
        write3(0xe5a0, r9, 0x2b, r12);
        //015e:2      mov    [op3 0xd666 r9 29 1], [op3 0xf49c r9 2 1]
        write3(0xd666, r9, 0x1d, read3(0xf49c, r9, 0x2));
        //0167:1      mov    [op3 0xc9c0 r9 12 1], [op3 0xe216 r9 12 1]
        write3(0xc9c0, r9, 0xc, read3(0xe216, r9, 0xc));
        //0170:0      not    [op3 0xc9c0 r9 12 1]
        write3(0xc9c0, r9, 0xc, ~read3(0xc9c0, r9, 0xc));
        //0175:2      or     [op3 0xd666 r9 29 1], [op3 0xc9c0 r9 12 1]
        write3(0xd666, r9, 0x1d, read3(0xd666, r9, 0x1d) | read3(0xc9c0, r9, 0xc));
        //017e:1      not    [op3 0xd666 r9 29 1]
        write3(0xd666, r9, 0x1d, ~read3(0xd666, r9, 0x1d));
        //0183:3      or     [op3 0xe5a0 r9 43 1], [op3 0xd666 r9 29 1]
        write3(0xe5a0, r9, 0x2b, read3(0xe5a0, r9, 0x2b) | read3(0xd666, r9, 0x1d));
        //018c:2      mov    [op3 0xea26 r9 49 1], [op3 0xe5a0 r9 43 1]
        write3(0xea26, r9, 0x31, read3(0xe5a0, r9, 0x2b));
        //0195:1      and    [op3 0xf49c r9 2 1], [op3 0xe216 r9 12 1]
```

```
        write3(0xf49c, r9, 0x2, read3(0xf49c, r9, 0x2) & read3(0xe216, r9, 0xc));
        //019e:0      mov    [op3 0xe216 r9 12 1], [op3 0xf49c r9 2 1]
        write3(0xe216, r9, 0xc, read3(0xf49c, r9, 0x2));
        //01a6:7      mov    [op3 0xf49c r9 2 1], [op3 0xea26 r9 49 1]
        write3(0xf49c, r9, 0x2, read3(0xea26, r9, 0x31));
        zz = read3(0xe216, r9, 0xc) & 0x8000;
        //01af:6      shlf0  [op3 0xe216 r9 12 1], 1
        write3(0xe216, r9, 0xc, read3(0xe216, r9, 0xc)<<1);
        if(zz){
        //01b6:1      orC0   [op3 0xd944 r9 13 1], 0xa1a6
        write3(0xd944, r9, 0xd, read3(0xd944, r9, 0xd) | 0xa1a6);
        }
//write3(0xe216, r9, 0xc, read3(0xe216, r9, 0xc) & read3(0xe216, r9, 0xc));

    //01c6:5      jNZ0   13d:6 abs
    }while(read3(0xe216, r9, 0xc));

    //01cb:6      not    [op3 0xd944 r9 13 1]
    write3(0xd944, r9, 0xd, ~read3(0xd944, r9, 0xd));
    //01d1:0      shlf0  [op3 0xd944 r9 13 1], 1
    write3(0xd944, r9, 0xd, read3(0xd944, r9, 0xd)<<1);
    //01d7:3      mov    r13, [op3 0xf49c r9 2 1]
    r13 = read3(0xf49c, r9, 0x2);
    //01dd:4      mov    [op3 0xea26 r9 49 1], 0x6d6d
    write3(0xea26, r9, 0x31, 0x6d6d);
    //01e5:1      mov    [op3 0xc82a r9 61 1], r13
    write3(0xc82a, r9, 0x3d, r13);
    //01eb:2      mov    [op3 0xf97a r9 15 1], [op3 0xdb2a r9 59 1]
    write3(0xf97a, r9, 0xf, read3(0xdb2a, r9, 0x3b));
    do
    {
        //01f4:1      mov    [op3 0xe38e r9 39 1], [op3 0xc82a r9 61 1]
        write3(0xe38e, r9, 0x27, read3(0xc82a, r9, 0x3d));
        //01fd:0      mov    [op3 0xc07c r9 17 1], [op3 0xf97a r9 15 1]
        write3(0xc07c, r9, 0x11, read3(0xf97a, r9, 0xf));
        //0205:7      not    [op3 0xc07c r9 17 1]
        write3(0xc07c, r9, 0x11, ~read3(0xc07c, r9, 0x11));
        //020b:1      and    [op3 0xe38e r9 39 1], [op3 0xc07c r9 17 1]
        write3(0xe38e, r9, 0x27, read3(0xe38e, r9, 0x27) & read3(0xc07c, r9, 0x11));
        //0214:0      mov    [op3 0xc07c r9 17 1], [op3 0xe38e r9 39 1]
        write3(0xc07c, r9, 0x11, read3(0xe38e, r9, 0x27));
        //021c:7      mov    [op3 0xdbc6 r9 36 1], [op3 0xc82a r9 61 1]
        write3(0xdbc6, r9, 0x24, read3(0xc82a, r9, 0x3d));
        //0225:6      mov    [op3 0xc76e r9 3 1], [op3 0xf97a r9 15 1]
        write3(0xc76e, r9, 0x3, read3(0xf97a, r9, 0xf));
        //022e:5      not    [op3 0xc76e r9 3 1]
        write3(0xc76e, r9, 0x3, ~read3(0xc76e, r9, 0x3));
        //0233:7      or     [op3 0xdbc6 r9 36 1], [op3 0xc76e r9 3 1]
        write3(0xdbc6, r9, 0x24, read3(0xdbc6, r9, 0x24) | read3(0xc76e, r9, 0x3));
        //023c:6      not    [op3 0xdbc6 r9 36 1]
        write3(0xdbc6, r9, 0x24, ~read3(0xdbc6, r9, 0x24));
        //0242:0      or     [op3 0xc07c r9 17 1], [op3 0xdbc6 r9 36 1]
        write3(0xc07c, r9, 0x11, read3(0xc07c, r9, 0x11) | read3(0xdbc6, r9, 0x24));
        //024a:7      mov    [op3 0xcb82 r9 14 1], [op3 0xc07c r9 17 1]
        write3(0xcb82, r9, 0xe, read3(0xc07c, r9, 0x11));
        //0253:6      and    [op3 0xc82a r9 61 1], [op3 0xf97a r9 15 1]
        write3(0xc82a, r9, 0x3d, read3(0xc82a, r9, 0x3d) & read3(0xf97a, r9, 0xf));
        //025c:5      mov    [op3 0xf97a r9 15 1], [op3 0xc82a r9 61 1]
        write3(0xf97a, r9, 0xf, read3(0xc82a, r9, 0x3d));
        //0265:4      mov    [op3 0xc82a r9 61 1], [op3 0xcb82 r9 14 1]
        write3(0xc82a, r9, 0x3d, read3(0xcb82, r9, 0xe));
        zz = read3(0xf97a, r9, 0xf) & 0x8000;
        //026e:3      shlf1  [op3 0xf97a r9 15 1], 1
        write3(0xf97a, r9, 0xf, read3(0xf97a, r9, 0xf)<<1);
        if(zz){
        //0274:6      orC1   [op3 0xea26 r9 49 1], 0xac93
        write3(0xea26, r9, 0x31, read3(0xea26, r9, 0x31) | 0xac93);
        }
//write3(0xf97a, r9, 0xf, read3(0xf97a, r9, 0xf) & read3(0xf97a, r9, 0xf));

    //0285:2      jNZ1   1f4:1 abs
    }while(read3(0xf97a, r9, 0xf));

    //028a:3      not    [op3 0xea26 r9 49 1]
    write3(0xea26, r9, 0x31, ~read3(0xea26, r9, 0x31));
    //028f:5      shlf1  [op3 0xea26 r9 49 1], 1
    write3(0xea26, r9, 0x31, read3(0xea26, r9, 0x31)<<1);
    //0296:0      mov    [op3 0xdb2a r9 59 1], [op3 0xc82a r9 61 1]
    write3(0xdb2a, r9, 0x3b, read3(0xc82a, r9, 0x3d));
    //029e:7      mov    [op3 0xea26 r9 49 1], 0x0
    write3(0xea26, r9, 0x31, 0x0);
    //02a6:4      mov    [op3 0xf9ca r9 59 1], r7
    write3(0xf9ca, r9, 0x3b, r7);
    //02ac:5      and    [op3 0xf9ca r9 59 1], r0
    write3(0xf9ca, r9, 0x3b, read3(0xf9ca, r9, 0x3b) & r0);
    //02b2:6      mov    r0, 0x4
    r0 = 0x4;
    do
```

```
    {
        //02b7:5      mov    [op3 0xd3d4 r9 38 1], 0x7703
        write3(0xd3d4, r9, 0x26, 0x7703);
        //02bf:2      shl    [op3 0xd3d4 r9 38 1], 1
        write3(0xd3d4, r9, 0x26, read3(0xd3d4, r9, 0x26)<<1);
        //02c5:5      mov    [op3 0xf7f8 r9 9 1], [op3 0xa00c r0 51 1]
        write3(0xf7f8, r9, 0x9, read3(0xa00c, r0, 0x33));
        //02ce:4      mov    [op3 0xc2d2 r9 16 1], 0x416e
        write3(0xc2d2, r9, 0x10, 0x416e);
        //02d6:1      mov    r12, [op3 0xf7f8 r9 9 1]
        r12 = read3(0xf7f8, r9, 0x9);
        //02dc:2      or     r12, [op3 0xdb2a r9 59 1]
        r12 |= read3(0xdb2a, r9, 0x3b);
        //02e2:3      not    r12
        r12 = ~r12;
        //02e4:7      not    r12
        r12 = ~r12;
        //02e7:3      mov    [op3 0xd5aa r9 32 1], [op3 0xf7f8 r9 9 1]
        write3(0xd5aa, r9, 0x20, read3(0xf7f8, r9, 0x9));
        //02f0:2      and    [op3 0xd5aa r9 32 1], [op3 0xdb2a r9 59 1]
        write3(0xd5aa, r9, 0x20, read3(0xd5aa, r9, 0x20) & read3(0xdb2a, r9, 0x3b));
        //02f9:1      not    [op3 0xd5aa r9 32 1]
        write3(0xd5aa, r9, 0x20, ~read3(0xd5aa, r9, 0x20));
        //02fe:3      and    r12, [op3 0xd5aa r9 32 1]
        r12 &= read3(0xd5aa, r9, 0x20);
        //0304:4      mov    [op3 0xf7f8 r9 9 1], r12
        write3(0xf7f8, r9, 0x9, r12);
        //030a:5      mov    r12, r8
        r12 = r8;
        //030e:0      and    r12, r0
        r12 &= r0;
        //0311:3      mov    1[0xa00c + r0], [op3 0xf7f8 r9 9 1]
        *((uint16_t*)&ram[0xa00c + r0]) = read3(0xf7f8, r9, 0x9);
        //0319:6      mov    [op3 0xd062 r9 54 1], r7
        write3(0xd062, r9, 0x36, r7);
        //031f:7      mov    [op3 0xec5a r9 45 1], 0x2619
        write3(0xec5a, r9, 0x2d, 0x2619);
        //0327:4      mov    [op3 0xc430 r9 60 1], 0x1
        write3(0xc430, r9, 0x3c, 0x1);
        //032f:1      mov    [op3 0xf850 r9 13 1], [op3 0xea26 r9 49 1]
        write3(0xf850, r9, 0xd, read3(0xea26, r9, 0x31));
        do
        {
            //0338:0      mov    r12, [op3 0xc430 r9 60 1]
            r12 = read3(0xc430, r9, 0x3c);
            //033e:1      and    r12, [op3 0xf850 r9 13 1]
            r12 &= read3(0xf850, r9, 0xd);
            //0344:2      not    r12
            r12 = ~r12;
            //0346:6      mov    [op3 0xd920 r9 7 1], [op3 0xc430 r9 60 1]
            write3(0xd920, r9, 0x7, read3(0xc430, r9, 0x3c));
            //034f:5      or     [op3 0xd920 r9 7 1], [op3 0xf850 r9 13 1]
            write3(0xd920, r9, 0x7, read3(0xd920, r9, 0x7) | read3(0xf850, r9, 0xd));
            //0358:4      not    [op3 0xd920 r9 7 1]
            write3(0xd920, r9, 0x7, ~read3(0xd920, r9, 0x7));
            //035d:6      not    [op3 0xd920 r9 7 1]
            write3(0xd920, r9, 0x7, ~read3(0xd920, r9, 0x7));
            //0363:0      and    r12, [op3 0xd920 r9 7 1]
            r12 &= read3(0xd920, r9, 0x7);
            //0369:1      mov    [op3 0xc99a r9 9 1], r12
            write3(0xc99a, r9, 0x9, r12);
            //036f:2      and    [op3 0xc430 r9 60 1], [op3 0xf850 r9 13 1]
            write3(0xc430, r9, 0x3c, read3(0xc430, r9, 0x3c) & read3(0xf850, r9, 0xd));
            //0378:1      mov    [op3 0xf850 r9 13 1], [op3 0xc430 r9 60 1]
            write3(0xf850, r9, 0xd, read3(0xc430, r9, 0x3c));
            //0381:0      mov    [op3 0xc430 r9 60 1], [op3 0xc99a r9 9 1]
            write3(0xc430, r9, 0x3c, read3(0xc99a, r9, 0x9));
            zz = read3(0xf850, r9, 0xd) & 0x8000;
            //0389:7      shlf1  [op3 0xf850 r9 13 1], 1
            write3(0xf850, r9, 0xd, read3(0xf850, r9, 0xd)<<1);
            if(zz)
            {
            //0390:2      orC1   [op3 0xec5a r9 45 1], 0x9994
            write3(0xec5a, r9, 0x2d, read3(0xec5a, r9, 0x2d) | 0x9994);
            }
//write3(0xf850, r9, 0xd, read3(0xf850, r9, 0xd) & read3(0xf850, r9, 0xd));

        //03a0:6      jNZ1   338:0 abs
        }while(read3(0xf850, r9, 0xd));

        //03a5:7      not    [op3 0xec5a r9 45 1]
        write3(0xec5a, r9, 0x2d, ~read3(0xec5a, r9, 0x2d));
        //03ab:1      shlf1  [op3 0xec5a r9 45 1], 1
        write3(0xec5a, r9, 0x2d, read3(0xec5a, r9, 0x2d)<<1);
        //03b1:4      mov    [op3 0xea26 r9 49 1], [op3 0xc430 r9 60 1]
        write3(0xea26, r9, 0x31, read3(0xc430, r9, 0x3c));
        //03ba:3      mov    [op3 0xec5a r9 45 1], r3
        write3(0xec5a, r9, 0x2d, r3);
```

```
        //03c0:4      or     [op3 0xec5a r9 45 1], r5
        write3(0xec5a, r9, 0x2d, read3(0xec5a, r9, 0x2d) | r5);
        //03c6:5      mov    [op3 0xcd96 r9 37 1], 0x7eb2
        write3(0xcd96, r9, 0x25, 0x7eb2);
        //03ce:2      mov    [op3 0xe292 r9 10 1], 0x2
        write3(0xe292, r9, 0xa, 0x2);
        //03d5:7      mov    [op3 0xd5a4 r9 46 1], r0
        write3(0xd5a4, r9, 0x2e, r0);
        do
        {
            //03dc:0      mov    [op3 0xd904 r9 17 1], [op3 0xe292 r9 10 1]
            write3(0xd904, r9, 0x11, read3(0xe292, r9, 0xa));
            //03e4:7      and    [op3 0xd904 r9 17 1], [op3 0xd5a4 r9 46 1]
            write3(0xd904, r9, 0x11, read3(0xd904, r9, 0x11) & read3(0xd5a4, r9, 0x2e));
            //03ed:6      not    [op3 0xd904 r9 17 1]
            write3(0xd904, r9, 0x11, ~read3(0xd904, r9, 0x11));
            //03f3:0      mov    r12, [op3 0xe292 r9 10 1]
            r12 = read3(0xe292, r9, 0xa);
            //03f9:1      or     r12, [op3 0xd5a4 r9 46 1]
            r12 |= read3(0xd5a4, r9, 0x2e);
            //03ff:2      not    r12
            r12 = ~r12;
            //0401:6      not    r12
            r12 = ~r12;
            //0404:2      and    [op3 0xd904 r9 17 1], r12
            write3(0xd904, r9, 0x11, read3(0xd904, r9, 0x11) & r12);
            //040a:3      mov    [op3 0xead6 r9 17 1], [op3 0xd904 r9 17 1]
            write3(0xead6, r9, 0x11, read3(0xd904, r9, 0x11));
            //0413:2      and    [op3 0xe292 r9 10 1], [op3 0xd5a4 r9 46 1]
            write3(0xe292, r9, 0xa, read3(0xe292, r9, 0xa) & read3(0xd5a4, r9, 0x2e));
            //041c:1      mov    [op3 0xd5a4 r9 46 1], [op3 0xe292 r9 10 1]
            write3(0xd5a4, r9, 0x2e, read3(0xe292, r9, 0xa));
            //0425:0      mov    [op3 0xe292 r9 10 1], [op3 0xead6 r9 17 1]
            write3(0xe292, r9, 0xa, read3(0xead6, r9, 0x11));
            zz = read3(0xd5a4, r9, 0x2e) & 0x8000;
            //042d:7      shlf1  [op3 0xd5a4 r9 46 1], 1
            write3(0xd5a4, r9, 0x2e, read3(0xd5a4, r9, 0x2e)<<1);
            if(zz)
            {
            //0434:2      orC1   [op3 0xcd96 r9 37 1], 0x8a22
            write3(0xcd96, r9, 0x25, read3(0xcd96, r9, 0x25) | 0x8a22);
            //write3(0xd5a4, r9, 0x2e, read3(0xd5a4, r9, 0x2e) & read3(0xd5a4, r9, 0x2e));
            }
        //0444:6      jNZ1   3dc:0 abs
        }while(read3(0xd5a4, r9, 0x2e));

        //0449:7      not    [op3 0xcd96 r9 37 1]
        write3(0xcd96, r9, 0x25, ~read3(0xcd96, r9, 0x25));
        //044f:1      shlf1  [op3 0xcd96 r9 37 1], 1
        write3(0xcd96, r9, 0x25, read3(0xcd96, r9, 0x25)<<1);
        //0455:4      mov    r0, [op3 0xe292 r9 10 1]
        r0 = read3(0xe292, r9, 0xa);
        //045b:5      mov    [op3 0xead6 r9 17 1], [op3 0xea26 r9 49 1]
        write3(0xead6, r9, 0x11, read3(0xea26, r9, 0x31));
        //0464:4      mov    [op3 0xd5a4 r9 46 1], 0x10
        write3(0xd5a4, r9, 0x2e, 0x10);
        //046c:1      not    [op3 0xd5a4 r9 46 1]
        write3(0xd5a4, r9, 0x2e, ~read3(0xd5a4, r9, 0x2e));
        //0471:3      mov    [op3 0xf106 r9 51 1], 0x684e
        write3(0xf106, r9, 0x33, 0x684e);
        //0479:0      mov    [op3 0xcec2 r9 59 1], 0x1
        write3(0xcec2, r9, 0x3b, 0x1);
        //0480:5      mov    [op3 0xe244 r9 51 1], [op3 0xd5a4 r9 46 1]
        write3(0xe244, r9, 0x33, read3(0xd5a4, r9, 0x2e));
        do
        {
            //0489:4      mov    [op3 0xe0fe r9 31 1], [op3 0xcec2 r9 59 1]
            write3(0xe0fe, r9, 0x1f, read3(0xcec2, r9, 0x3b));
            //0492:3      mov    [op3 0xd5ae r9 51 1], [op3 0xe244 r9 51 1]
            write3(0xd5ae, r9, 0x33, read3(0xe244, r9, 0x33));
            //049b:2      not    [op3 0xd5ae r9 51 1]
            write3(0xd5ae, r9, 0x33, ~read3(0xd5ae, r9, 0x33));
            //04a0:4      or     [op3 0xe0fe r9 31 1], [op3 0xd5ae r9 51 1]
            write3(0xe0fe, r9, 0x1f, read3(0xe0fe, r9, 0x1f) | read3(0xd5ae, r9, 0x33));
            //04a9:3      not    [op3 0xe0fe r9 31 1]
            write3(0xe0fe, r9, 0x1f, ~read3(0xe0fe, r9, 0x1f));
            //04ae:5      mov    [op3 0xec46 r9 40 1], [op3 0xcec2 r9 59 1]
            write3(0xec46, r9, 0x28, read3(0xcec2, r9, 0x3b));
            //04b7:4      mov    [op3 0xe388 r9 47 1], [op3 0xe244 r9 51 1]
            write3(0xe388, r9, 0x2f, read3(0xe244, r9, 0x33));
            //04c0:3      not    [op3 0xe388 r9 47 1]
            write3(0xe388, r9, 0x2f, ~read3(0xe388, r9, 0x2f));
            //04c5:5      and    [op3 0xec46 r9 40 1], [op3 0xe388 r9 47 1]
            write3(0xec46, r9, 0x28, read3(0xec46, r9, 0x28) & read3(0xe388, r9, 0x2f));
            //04ce:4      or     [op3 0xe0fe r9 31 1], [op3 0xec46 r9 40 1]
            write3(0xe0fe, r9, 0x1f, read3(0xe0fe, r9, 0x1f) | read3(0xec46, r9, 0x28));
            //04d7:3      mov    [op3 0xf2bc r9 53 1], [op3 0xe0fe r9 31 1]
            write3(0xf2bc, r9, 0x35, read3(0xe0fe, r9, 0x1f));
```

```
            //04e0:2      and    [op3 0xcec2 r9 59 1], [op3 0xe244 r9 51 1]
            write3(0xcec2, r9, 0x3b, read3(0xcec2, r9, 0x3b) & read3(0xe244, r9, 0x33));
            //04e9:1      mov    [op3 0xe244 r9 51 1], [op3 0xcec2 r9 59 1]
            write3(0xe244, r9, 0x33, read3(0xcec2, r9, 0x3b));
            //04f2:0      mov    [op3 0xcec2 r9 59 1], [op3 0xf2bc r9 53 1]
            write3(0xcec2, r9, 0x3b, read3(0xf2bc, r9, 0x35));
            zz = read3(0xe244, r9, 0x33) & 0x8000;
            //04fa:7      shlf1  [op3 0xe244 r9 51 1], 1
            write3(0xe244, r9, 0x33, read3(0xe244, r9, 0x33)<<1);
            if(zz)
            //0501:2      orC1   [op3 0xf106 r9 51 1], 0xc433
            {write3(0xf106, r9, 0x33, read3(0xf106, r9, 0x33) | 0xc433);
            }
//write3(0xe244, r9, 0x33, read3(0xe244, r9, 0x33) & read3(0xe244, r9, 0x33));

        //0511:6      jNZ1   489:4 abs
        }while(read3(0xe244, r9, 0x33));
        //0516:7      not    [op3 0xf106 r9 51 1]
        write3(0xf106, r9, 0x33, ~read3(0xf106, r9, 0x33));
        //051c:1      shlf1  [op3 0xf106 r9 51 1], 1
        write3(0xf106, r9, 0x33, read3(0xf106, r9, 0x33)<<1);
        //0522:4      mov    [op3 0xd5a4 r9 46 1], [op3 0xcec2 r9 59 1]
        write3(0xd5a4, r9, 0x2e, read3(0xcec2, r9, 0x3b));
        //052b:3      mov    [op3 0xe1a6 r9 11 1], 0x2930
        write3(0xe1a6, r9, 0xb, 0x2930);
        //0533:0      mov    [op3 0xda82 r9 36 1], [op3 0xead6 r9 17 1]
        write3(0xda82, r9, 0x24, read3(0xead6, r9, 0x11));
        //053b:7      mov    [op3 0xe20e r9 55 1], [op3 0xd5a4 r9 46 1]
        write3(0xe20e, r9, 0x37, read3(0xd5a4, r9, 0x2e));
        do
        {
            //0544:6      mov    [op3 0xd5f8 r9 37 1], [op3 0xda82 r9 36 1]
            write3(0xd5f8, r9, 0x25, read3(0xda82, r9, 0x24));
            //054d:5      not    [op3 0xd5f8 r9 37 1]
            write3(0xd5f8, r9, 0x25, ~read3(0xd5f8, r9, 0x25));
            //0552:7      and    [op3 0xd5f8 r9 37 1], [op3 0xe20e r9 55 1]
            write3(0xd5f8, r9, 0x25, read3(0xd5f8, r9, 0x25) & read3(0xe20e, r9, 0x37));
            //055b:6      mov    [op3 0xcb10 r9 22 1], [op3 0xda82 r9 36 1]
            write3(0xcb10, r9, 0x16, read3(0xda82, r9, 0x24));
            //0564:5      mov    [op3 0xebb4 r9 60 1], [op3 0xe20e r9 55 1]
            write3(0xebb4, r9, 0x3c, read3(0xe20e, r9, 0x37));
            //056d:4      not    [op3 0xebb4 r9 60 1]
            write3(0xebb4, r9, 0x3c, ~read3(0xebb4, r9, 0x3c));
            //0572:6      and    [op3 0xcb10 r9 22 1], [op3 0xebb4 r9 60 1]
            write3(0xcb10, r9, 0x16, read3(0xcb10, r9, 0x16) & read3(0xebb4, r9, 0x3c));
            //057b:5      or     [op3 0xd5f8 r9 37 1], [op3 0xcb10 r9 22 1]
            write3(0xd5f8, r9, 0x25, read3(0xd5f8, r9, 0x25) | read3(0xcb10, r9, 0x16));
            //0584:4      mov    [op3 0xc450 r9 32 1], [op3 0xd5f8 r9 37 1]
            write3(0xc450, r9, 0x20, read3(0xd5f8, r9, 0x25));
            //058d:3      and    [op3 0xda82 r9 36 1], [op3 0xe20e r9 55 1]
            write3(0xda82, r9, 0x24, read3(0xda82, r9, 0x24) & read3(0xe20e, r9, 0x37));
            //0596:2      mov    [op3 0xe20e r9 55 1], [op3 0xda82 r9 36 1]
            write3(0xe20e, r9, 0x37, read3(0xda82, r9, 0x24));
            //059f:1      mov    [op3 0xda82 r9 36 1], [op3 0xc450 r9 32 1]
            write3(0xda82, r9, 0x24, read3(0xc450, r9, 0x20));
            zz = read3(0xe20e, r9, 0x37) & 0x8000;
            //05a8:0      shlf0  [op3 0xe20e r9 55 1], 1
            write3(0xe20e, r9, 0x37, read3(0xe20e, r9, 0x37)<<1);
            if(zz){
            //05ae:3      orC0   [op3 0xe1a6 r9 11 1], 0xb929
            write3(0xe1a6, r9, 0xb, read3(0xe1a6, r9, 0xb) | 0xb929);
            //write3(0xe20e, r9, 0x37, read3(0xe20e, r9, 0x37) & read3(0xe20e, r9, 0x37));
            }
        //05be:7      jNZ0   544:6 abs
        }while(read3(0xe20e, r9, 0x37));
        //05c4:0      not    [op3 0xe1a6 r9 11 1]
        write3(0xe1a6, r9, 0xb, ~read3(0xe1a6, r9, 0xb));
        zz = read3(0xe1a6, r9, 0xb) & 0x8000;
        //05c9:2      shlf0  [op3 0xe1a6 r9 11 1], 1
        write3(0xe1a6, r9, 0xb, read3(0xe1a6, r9, 0xb)<<1);
        //05cf:5      mov    [op3 0xead6 r9 17 1], [op3 0xda82 r9 36 1]
        write3(0xead6, r9, 0x11, read3(0xda82, r9, 0x24));
        //05d8:4      andf0  [op3 0xead6 r9 17 1], [op3 0xead6 r9 17 1]
        write3(0xead6, r9, 0x11, read3(0xead6, r9, 0x11) & read3(0xead6, r9, 0x11));
    //05e1:3      jC0    2b7:5 abs
    }while(zz);

    //05e6:4      mov    [0x8000], 0xdead
    *((uint16_t*)&ram[0x8000]) = 0xdead;
    //05ed:1      mov    [op3 0xcac6 r9 52 1], r1
    write3(0xcac6, r9, 0x34, r1);
    //05f3:2      and    [op3 0xcac6 r9 52 1], r8
    write3(0xcac6, r9, 0x34, read3(0xcac6, r9, 0x34) & r8);
    //05f9:3      mov    [0x8002], 0xbeef
    *((uint16_t*)&ram[0x8002]) = 0xbeef;

    return !memcmp(&ram[0xa010], "V29vdCAhISBTbWVsbHMgZ29vZCA6KQ==", 32);
}
```

```
559.
560. int main(int argc, char** argv)
561. {
562.     uint16_t i;
563.
564.     for(i=0xffff; i > 0 ; i--)
565.     {
566.         if(check_key3(0xf63d, i))
567.         {
568.             printf("Key? 0x%x 0x%x\n", 0xf63d, i);
569.             printf("%s\n", &ram[0xa010]);
570.             break;
571.         }
572.     }
573.     return 0;
574. }
```

SSTICDECRYPT.PY

```
1.  from Crypto.Cipher import ARC4
2.  import hashlib
3.  import sys
4.
5.  if len(sys.argv) != 3:
6.      print "Usage : %s file key" % sys.argv[0]
7.      exit(0)
8.
9.  f = open(sys.argv[1], "rb")
10. check = f.read(16)
11. data = f.read()
12. f.close()
13.
14. if hashlib.md5(data).digest() == check:
15.     print "MD5 check OK"
16. else:
17.     print "MD5 check fail"
18.
19. data = map(ord, data)
20. for i in xrange(1, len(data)-1):
21.     data[i-1] = (data[i-1] ^ data[i])
22.
23. data = ARC4.new(sys.argv[2]).decrypt("".join(map(chr,data)))
24.
25. outputfile = sys.argv[1] + ".dec"
26. print "Writing file %s" % outputfile
27. f = open(outputfile, "wb")
28. f.write(data)
29. f.close()
```