Aluno: José Victor Medeiros Thomé da Silva
Instrutor: Leonardo Ventura

TAG – Engenharia Reversa

O que o programa da TAG faz?

Visualizando sua execução, o programa abre um diretório com o mesmo nome da pasta home do usuário (ou cria o mesmo vazio caso não exista) e criptografa todos os arquivos que tiverem contidos nele, deletando em seguida os arquivos originais.

Como o programa funciona?

Ao revertermos o executável para a linguagem assembly usando no GNU/Linux o comando `objdump --disassemble tag > tag.s`, além de diversas funções de controle relacionadas a operações feitas no hardware, vemos quatro funções notáveis: a `main`, `encripta_arquivos`, `_system_integrity_check`,`_system_loader_callback`. Elas, considerando como o programa funciona, são as principais responsáveis por todo o processo envolvido na encriptação dos arquivos criados.

Na `main`, como em qualquer código em C, é inicializado o programa. É nela que é feita a criação do diretório com o mesmo nome da pasta de usuário, assim como também são executadas a função `encripta_arquivos` para inicializar o gerador de números aleatórios, a `_system_integrity_check` para gerar a chave de encriptação e a `_system_loader_callback` para executar o encriptador baixado de uma URL. Abaixo temos o assembly da main.

```
00000000000015f1 <main>:
    15f1:  55                    push rbp #Salva o registro de ativação da função anterior.
    15f2:  48 89 e5              mov  rbp,rsp #Cria um novo registro de ativação.
    15f5:  48 83 ec 10           sub  rsp,0x10 #Abre 4 espaços de memória para o registro de ativação.
    15f9:  48 8d 3d 60 0a 00 00  lea  rdi,[rip+0xa60] #Pega o endereço da mensagem “Olá!” como parâmetro.
    1600:  e8 1b fb ff ff        call 1120 <puts@plt> #Imprime a mensagem no terminal.
    1605:  48 8d 3d 5c 0a 00 00  lea  rdi,[rip+0xa5c] #Pega como parâmetro a string de comando:
                                                      “mkdir -p $USER && cp ~/* $USER 2> /dev/null” para criar um
                                                      diretório vazio com o mesmo nome da pasta do usuário.
    160c:  e8 5f fa ff ff        call 1070 <system@plt> #Executa o comando “mkdir -p $USER && cp ~/*
                                                        $USER 2> /dev/null”.
    1611:  48 8d 3d 80 0a 00 00  lea  rdi,[rip+0xa80] #Pega o parâmetro “Codificando os arquivos da sua home...”
    1618:  e8 03 fb ff ff        call 1120 <puts@plt> #Imprime a mensagem no terminal.
    161d:  48 8d 3d 9c 0a 00 00  lea  rdi,[rip+0xa9c] #Pega o parâmetro “Procure por uma forma de descodificá-los”
    1624:  e8 f7 fa ff ff        call 1120 <puts@plt> #Imprime a mensagem no terminal.
    1629:  48 8d 3d c0 0a 00 00  lea   rdi,[rip+0xac0] #Pega o parâmetro “OBS: Não desligue sua máquina, se não
                                                       não será mais possível recuperar os dados!!!”
    1630:  e8 eb fa ff ff        call 1120 <puts@plt> #Imprime a mensagem no terminal.
    1635:  bf 01 00 00 00        mov  edi,0x1 #Pega 1 segundo como parâmetro de sleep().
    163a:  e8 21 fa ff ff        call 1060 <sleep@plt> #Chama a função sleep(1).
    163f:  b8 00 00 00 00        mov  eax,0x0
    1644:  e8 e6 fd ff ff        call 142f <encripta_arquivos> #Chama a função encripta_arquivos.
    1649:  b8 00 00 00 00        mov  eax,0x0
    164e:  e8 bd fd ff ff        call 1410 <printa_ascii_art> #Chama a função para printar a ascii_art.
    1653:  b8 00 00 00 00        mov  eax,0x0
    1658:  e8 56 fe ff ff        call  14b3 <_system_integrity_check> #Chama a função _system_integrity_check
                                                                       (que gera a chave de encriptação).
    165d:  89 45 fc              mov  DWORD PTR [rbp-0x4],eax #Guarda essa chave na pilha.
    1660:  8b 45 fc              mov  eax,DWORD PTR [rbp-0x4] #Recupera essa chave.
    1663:  89 c6                 mov  esi,eax #A coloca como segundo parâmetro  de _system_loader_callback.
    1665:  48 8d 3d dd 0a 00 00  lea  rdi,[rip+0xadd] #Pega a URL do encriptador como parâmetro.
```

```
166c: e8 be fe ff ff          call   152f <_system_loader_callback> #Chama __system_loader_callback para
                                                                    executar o encriptador.
1671: 48 8d 3d e8 0a 00 00    lea    rdi,[rip+0xae8] #Pega o parâmetro "brincadeira, fiz uma cópia da sua home
                                                     no diretório atual e encriptei seus arquivos lá, rs"
1678: e8 a3 fa ff ff          call   1120 <puts@plt> #Imprime a mensagem no terminal.
167d: b8 00 00 00 00          mov    eax,0x0 #Carrega o valor de retorno da rotina main (0 ou nada).
1682: c9                      leave  #Volta para a base da pilha para descartar o registro de ativação atual e
                                      recuperar o da função anterior.
1683: c3                      ret    #Restaura o endereço de retorno da sub-rotina que fez a chamada.
```

A função `encripta_arquivos`, apesar do nome, sua única função de fato é inicializar o gerador de números aleatórios a ser usado para gerar a chave criada na função `_system_integrity_check`, ou seja, ela inicializa `srand()` usando como parâmetro um valor de tempo retornado por `time((time_t *)0)`, que seria o mesmo que `srand((uint) time((time_t *)0))`. Abaixo está o assembly de `encripta_arquivos()`.

```
000000000000142f <encripta_arquivos>:
142f:  55                     push   rbp #Salva o registro de ativação da função anterior.
1430: 48 89 e5                mov    rbp,rsp #Cria um novo registro de ativação.
1433: 48 83 ec 20             sub    rsp,0x20 #Abre 8 espaços na pilha para o registro de ativação.
1437: bf 00 00 00 00          mov    edi,0x0 #Coloca 0 como parâmetro para a função time().
143c: e8 3f fc ff ff          call   1080 <time@plt> #Chama time().
1441: 89 c7                   mov    edi,eax #Recupera o valor de retorno (no caso o tempo).
1443: e8 48 fc ff ff          call   1090 <srand@plt> #Chama srand() para iniciar o gerador de números
                                                      aleatórios.
1448: e8 03 fc ff ff          call   1050 <rand@plt> #Gera um número aleatório.
144d: 89 c2                   mov    edx,eax
144f:  48 63 c2               movsxd rax,edx
1452: 48 69 c0 67 66 66 66    imul   rax,rax,0x66666667
1459: 48 c1 e8 20             shr    rax,0x20
145d: 89 c1                   mov    ecx,eax
145f:  c1 f9 02               sar    ecx,0x2
1462: 89 d0                   mov    eax,edx
1464: c1 f8 1f                sar    eax,0x1f
1467: 29 c1                   sub    ecx,eax
1469: 89 c8                   mov    eax,ecx
146b: 89 45 e8                mov    DWORD PTR [rbp-0x18],eax
146e: 8b 4d e8                mov    ecx,DWORD PTR [rbp-0x18]
1471: 89 c8                   mov    eax,ecx
1473: c1 e0 02                shl    eax,0x2
1476: 01 c8                   add    eax,ecx
1478: 01 c0                   add    eax,eax
147a: 29 c2                   sub    edx,eax
147c: 89 d0                   mov    eax,edx
147e: 89 45 e8                mov    DWORD PTR [rbp-0x18],eax
1481: 83 7d e8 04             cmp    DWORD PTR [rbp-0x18],0x4
1485: 7f 17                   jg     149e <encripta_arquivos+0x6f>
1487: c7 45 f4 0a 00 00 00    mov    DWORD PTR [rbp-0xc],0xa
148e: c7 45 f8 0f 00 00 00    mov    DWORD PTR [rbp-0x8],0xf
1495: c7 45 fc 19 00 00 00    mov    DWORD PTR [rbp-0x4],0x19
149c: eb 12                   jmp    14b0 <encripta_arquivos+0x81>
149e: c7 45 ec 41 00 00 00    mov    DWORD PTR [rbp-0x14],0x41
14a5: 8b 55 ec                mov    edx,DWORD PTR [rbp-0x14]
14a8: 8b 45 e8                mov    eax,DWORD PTR [rbp-0x18]
14ab: 01 d0                   add    eax,edx
14ad: 89 45 f0                mov    DWORD PTR [rbp-0x10],eax
14b0: 90                      nop
```

```
14b1: c9                    leave #Volta para a base da pilha para descartar o registro de ativação atual e
                                  recuperar o da função anterior.
14b2: c3                    ret #Restaura o endereço de retorno da rotina que fez a chamada (no caso a main).
```

A função `_system_integrity_check(void)` é responsável por gerar a chave para criptografar os arquivos, assim como armazenar a mesma dentro de um arquivo temporário na pasta "`/tmp`" do sistema. No caso a chave de encriptação é `<valor_aleatório> % 5 + 1`. Abaixo está o Assembly dessa função com a descrição dos seus comandos.

```
00000000000014b3 <_system_integrity_check>:
   14b3: 55                 push rbp #Salva o registro de ativação da função anterior.
   14b4: 48 89 e5           mov  rbp,rsp #Cria um novo registro de ativação.
   14b7: 48 83 ec 10        sub  rsp,0x10 #Abre 4 espaços na memória para o registro de ativação.
   14bb: e8 90 fb ff ff     call 1050 <rand@plt> #Gera um valor aleatório com rand().
   14c0: 89 c1              mov  ecx,eax #Recupera o número aleatório de retorno.
```

======= Nesse trecho abaixo é calculada a chave (valor_aleatorio_gerado (mod 5) + 1). =======

```
   14c2: 48 63 c1              movsxd rax,ecx
   14c5: 48 69 c0 67 66 66 66  imul   rax,rax,0x66666667
   14cc: 48 c1 e8 20           shr    rax,0x20
   14d0: 89 c2                 mov    edx,eax
   14d2: d1 fa                 sar    edx,1
   14d4: 89 c8                 mov    eax,ecx
   14d6: c1 f8 1f              sar    eax,0x1f
   14d9: 29 c2                 sub    edx,eax
   14db: 89 d0                 mov    eax,edx
   14dd: c1 e0 02              shl    eax,0x2
   14e0: 01 d0                 add    eax,edx
   14e2: 29 c1                 sub    ecx,eax
   14e4: 89 ca                 mov    edx,ecx
   14e6: 8d 42 01              lea    eax,[rdx+0x1]
```

==========

```
   14e9: 89 45 f4              mov  DWORD PTR [rbp-0xc],eax # Salva o valor da chave calculada na pilha.
   14ec: 48 8d 35 1b 0b 00 00  lea  rsi,[rip+0xb1b] #Pega "/tmp/key" como primeiro parâmetro.
   14f3: 48 8d 3d 17 0b 00 00  lea  rdi,[rip+0xb17] #Pega "w+" como segundo parâmetro.
   14fa: e8 d1 fb ff ff        call 10d0 <fopen@plt> #Abre "/tmp/key" para escrita.
   14ff: 48 89 45 f8           mov  QWORD PTR [rbp-0x8],rax # Salva na pilha a referência de "/tmp/key".
   1503: 8b 55 f4              mov  edx,DWORD PTR [rbp-0xc] # Recupera o valor da chave calculada.
   1506: 48 8b 45 f8           mov  rax,QWORD PTR [rbp-0x8] # Recupera a referência de "/tmp/key".
   150a: 48 8d 35 09 0b 00 00  lea  rsi,[rip+0xb09] #Pega a string de formatação "%d\n" como segundo parâmetro.
   1511: 48 89 c7              mov  rdi,rax # # Passa como primeiro parâmetro a referência de "/tmp/key".
   1514: b8 00 00 00 00        mov  eax,0x0 #
   1519: e8 f2 fb ff ff        call 1110 <fprintf@plt> #Insere a chave no arquivo "/tmp/key".
   151e: 48 8b 45 f8           mov  rax,QWORD PTR [rbp-0x8] # Recupera a referência de "/tmp/key".
   1522: 48 89 c7              mov  rdi,rax # Passa essa referência como parâmetro para fclose().
   1525: e8 76 fb ff ff        call 10a0 <fclose@plt> # Chama fclose() para fechar o arquivo.
   152a: 8b 45 f4              mov  eax,DWORD PTR [rbp-0xc] # Prepara a chave calculada para ser retornada.
   152d: c9                    leave
   152e: c3                    ret
```

Essa função, `_system_loader_callback(url_encrip, key)`, baixa da internet o código do encriptador localizado na url passada como primeiro parâmetro e o executa com a chave passada no segundo parâmetro. Esse encriptador, que é baixado da internet, pega cada arquivo do diretório criado com o nome do usuário e criptografa o conteúdo do mesmo somando os caracteres com o valor da chave, que é um método de criptografia simétrica. Abaixo está o Assembly de `__system_loader_callback()` com a descrição dos comandos mais importantes.

```
000000000000152f <_system_loader_callback>:
    152f:  55                      push   rbp
    1530: 48 89 e5                 mov    rbp,rsp
    1533: 48 81 ec c0 00 00 00     sub    rsp,0xc0
    153a: 48 89 bd 48 ff ff ff     mov    QWORD PTR [rbp-0xb8],rdi # Salva na pilha o primeiro parâmetro (a url).
    1541: 89 b5 44 ff ff ff        mov    DWORD PTR [rbp-0xbc],esi # Salva na pilha o segundo parâmetro (a chave).
    1547: 64 48 8b 04 25 28 00     mov    rax,QWORD PTR fs:0x28
    154e: 00 00
    1550: 48 89 45 f8              mov    QWORD PTR [rbp-0x8],rax
    1554: 31 c0                    xor    eax,eax
    1556: 48 8d 05 c1 0a 00 00     lea    rax,[rip+0xac1]
    155d: 48 89 85 58 ff ff ff     mov    QWORD PTR [rbp-0xa8],rax
    1564: 48 8d 05 b6 0a 00 00     lea    rax,[rip+0xab6]              # Pega o endereço da string ".encriptador"
    156b: 48 89 85 60 ff ff ff     mov    QWORD PTR [rbp-0xa0],rax # Salva o endereço de ".encriptador" na pilha.
    1572: 48 8b 95 60 ff ff ff     mov    rdx,QWORD PTR [rbp-0xa0] # Pega a string ".encriptador" como terceiro
                                                                   parâmetro de download_file_from_url().
    1579: 48 8b 85 48 ff ff ff     mov    rax,QWORD PTR [rbp-0xb8] # Pega a "url" do arquivo a ser baixado na pilha.
    1580: 48 89 d6                 mov    rsi,rdx # Passa ".encriptador" como segundo parâmetro.
    1583: 48 89 c7                 mov    rdi,rax # Passa a "url" como primeiro parâmetro.
    1586: e8 ba fd ff ff           call   1345 <download_file_from_url> # Chama download_file_from_url com os três
                                                                        parâmetros para baixar o encriptador.
    158b: 48 8d 05 9e 0a 00 00     lea    rax,[rip+0xa9e] # Pega o endereço da string do comando de execução para o
                                                          encriptador.
    1592: 48 89 85 68 ff ff ff     mov    QWORD PTR [rbp-0x98],rax # Guarda o endereço da string de comando na
                                                                   pilha.
    1599: 8b 8d 44 ff ff ff        mov    ecx,DWORD PTR [rbp-0xbc] # Pega a chave de encriptação da pilha como
                                                                   quarto parâmetro em sprintf.
    159f:  48 8b 95 68 ff ff ff    mov    rdx,QWORD PTR [rbp-0x98] # Pega da pilha o endereço da string de
                                                                   comando como segundo parâmetro em sprinf().
    15a6: 48 8d 85 70 ff ff ff     lea    rax,[rbp-0x90]  # Calcula o endereço cabeça de um buffer da memória a ser
                                                          usado em sprintf().
    15ad: 48 8d 35 a5 0a 00 00     lea    rsi,[rip+0xaa5]  # Pega o endereço de "%s %d\n" como segundo parâmetro
                                                           de sprintf().
    15b4: 48 89 c7                 mov    rdi,rax  # Pega o buffer como primeiro parâmetro em  sprintf().
    15b7: b8 00 00 00 00           mov    eax,0x0
    15bc: e8 7f fa ff ff           call   1040 <sprintf@plt>  # Chama sprintf() com seus parâmetros para construir o
                                                              comando de execução do encriptador.
    15c1: 48 8d 85 70 ff ff ff     lea    rax,[rbp-0x90] # Pega o endereço do buffer com o comando de execução do
                                                         encriptador completo.
    15c8: 48 89 c7                 mov    rdi,rax # Coloca em rdi como primeiro parâmetro em system().
    15cb: e8 a0 fa ff ff           call   1070 <system@plt> # Executa o encriptador através da função system().
    15d0: bf 02 00 00 00           mov    edi,0x2 # Armazena 2 em edi como parâmetro de sleep().
    15d5: e8 86 fa ff ff           call   1060 <sleep@plt> # Para a execução do processo por 2 segundos.
    15da: 90                       nop
    15db: 48 8b 45 f8              mov    rax,QWORD PTR [rbp-0x8]
    15df:  64 48 33 04 25 28 00    xor    rax,QWORD PTR fs:0x28
    15e6: 00 00
    15e8: 74 05                    je     15ef <_system_loader_callback+0xc0>
    15ea: e8 c1 fa ff ff           call   10b0 <__stack_chk_fail@plt>
    15ef:  c9                      leave
    15f0:  c3                      ret
```

Para descriptografar os arquivos, foi criado um outro programa (o descriptador) que recupera a chave de encriptação no arquivo "/tmp/key", soma os caracteres dos arquivos criptografados por essa mesma chave e salva o conteúdo descriptografado em novos arquivos.