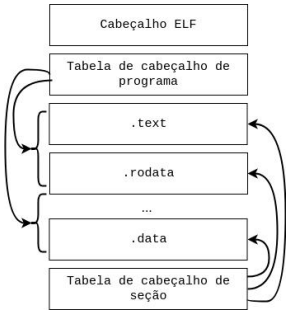
PROCESSO SELETIVO - Grupo de Resposta a Incidentes de Segurança

Nome: Fernanda Veiga Gomes da Fonseca
TAG - Engenharia Reversa - Entrega: 04/03/2020

Um arquivo ELF é um padrão comum de arquivo para executáveis. Antes de pesquisar mais sobre esse tipo de arquivo, através do terminal, foi acessado o diretório em que o arquivo foi salvo e o primeiro comando utilizado foi “cat tag”. Em uma análise rápida, foi possível identificar uma referência a encriptação de dados do diretório “home” do usuário e a uma cópia dos mesmos. De maneira a coletar mais informações sobre o arquivo, foram encontrados outros comandos que poderiam vir a ser úteis: “readelf” e “objdump”.

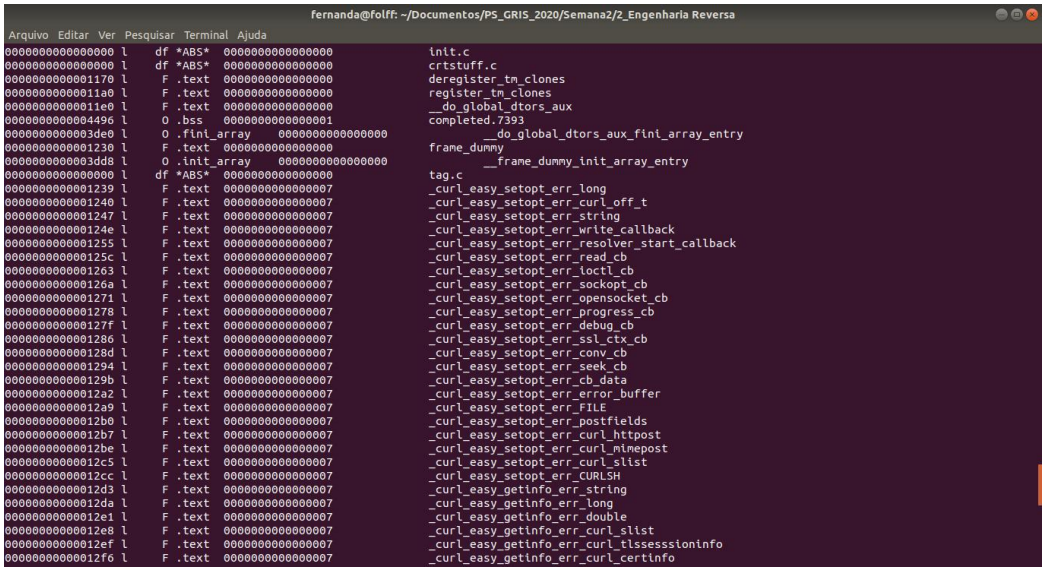
Dado o layout do arquivo, utilizou-se primeiramente o comando “objdump” para análise do arquivo, uma vez que possui opções para mostrar informação do cabeçalho, conteúdo do cabeçalho geral do arquivo, conteúdo dos cabeçalhos de seção e conteúdo assembler de todas as seções, por exemplo.



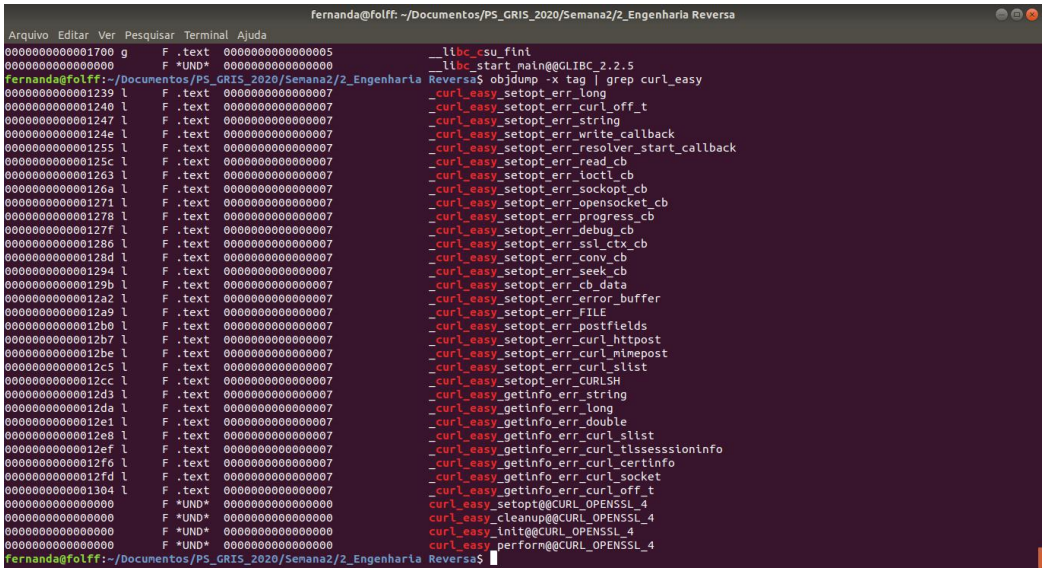

1- “objdump -a tag”
tag: formato do arquivo elf64-x86-64

2- “objdump -x tag”
Utilizando esse comando, foram identificados diversos arquivos .txt com nomes começando da mesma forma “_curl_easy_”*.

*“libcurl” é uma biblioteca para transferência de URLs gratuita e fácil de usar no lado do cliente. Ao usar a interface "easy" do libcurl, é iniciada uma sessão com um identificador, usado como entrada para as funções da interface.*




Para identificar os usos dessa biblioteca, foi utilizado o comando “objdump -x tag | grep curl_easy”.




Com ele, foram percebidos os usos de “curl_easy_setopt” e “curl_easy_getinfo”.

a) “curl_easy_setopt”: define opções da transferência e alguns retornos de chamada quando houver dados disponíveis (exemplos: URL e opções de erro). No caso do executável, o endereço é http://ix.io/2c6V.

b) “curl_easy_getinfo”: solicita informações internas da sessão curl.

Em seguida, há uma sequência de chamadas de funções.

a) “curl_easy_cleanup”: última função a ser chamada com o “easy_handle” da entrada (identificador) que a chamada “curl_easy_init” retornou.

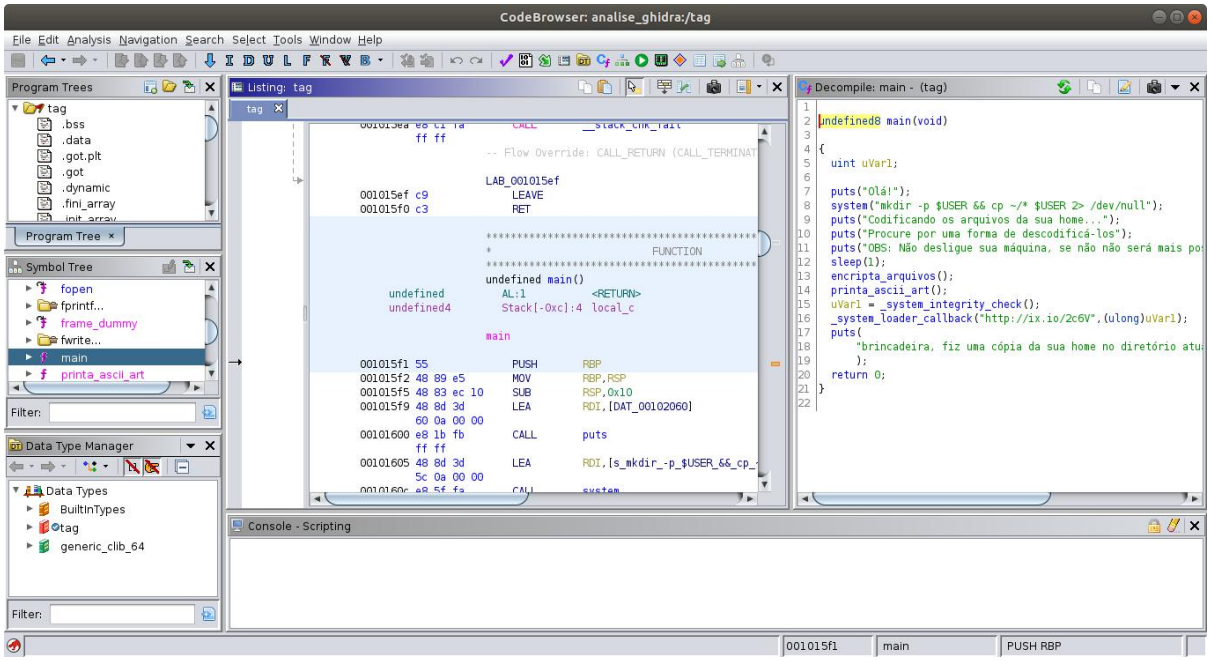
b) “curl_easy_init”: primeira função a ser chamada. Retorna um identificador CURL que deve ser usado como entrada para outras funções na interface.

c) “curl_easy_perform”: função chamada após as chamadas “curl_easy_init” e “curl_easy_setopt” para a execução da transferência descrita nas opções, devendo ser usado o mesmo identificador retornado por “curl_easy_init”.

3- “objdump -d tag”

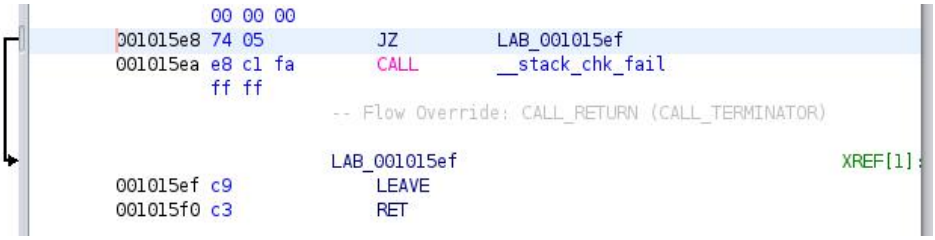
Esse comando permite a visualização do código Assembly das seções executáveis. Entretanto, o padrão utilizado é AT&T. Dessa forma, optou-se por utilizar o software Ghidra para visualizar o código no padrão Intel e, ainda, o mesmo na linguagem C.

Com o Ghidra, foi possível visualizar as linhas de código da função “main”.

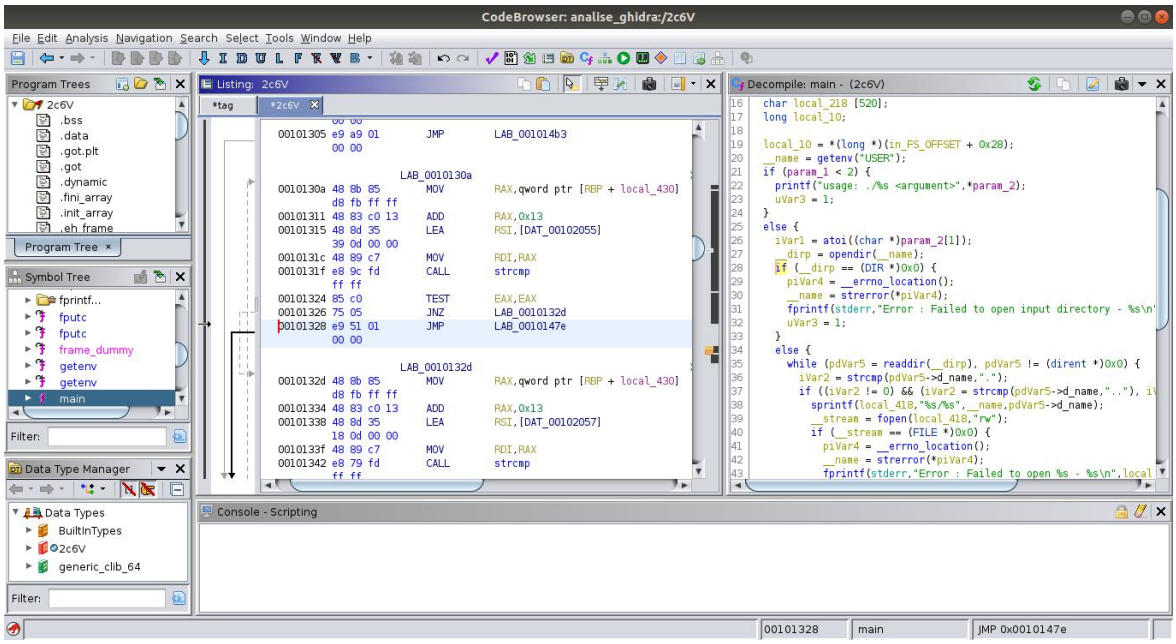



A análise da função revela que a chamada de outra função (“_system_loader_callback”) é a principal no contexto do executável. Ela é responsável por baixar o arquivo usado na encriptação do diretório através da chamada da função “download_file_from_url”, que utiliza a biblioteca libcurl para criar uma interface para a transferência.

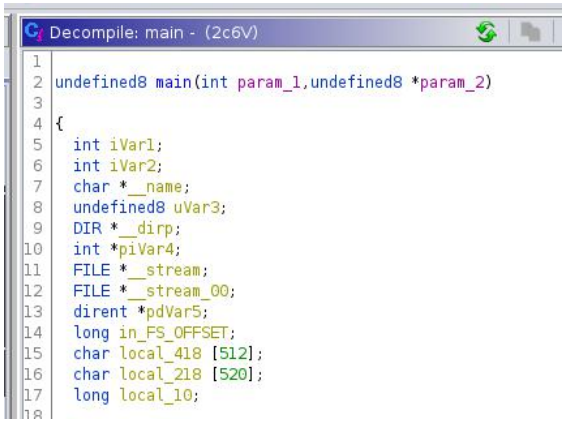
Em seguida, a função “sprintf” é chamada recursivamente para que cada item do diretório seja encriptado usando o arquivo baixado. A escrita utiliza um buffer e, caso seja constatado que as entradas ultrapassam um dado limite, indica um erro (não está ocorrendo retorno da subrotina) e chama outra função, deixando de retornar na função original. Por isso, a existência de um comando JZ em Assembly.



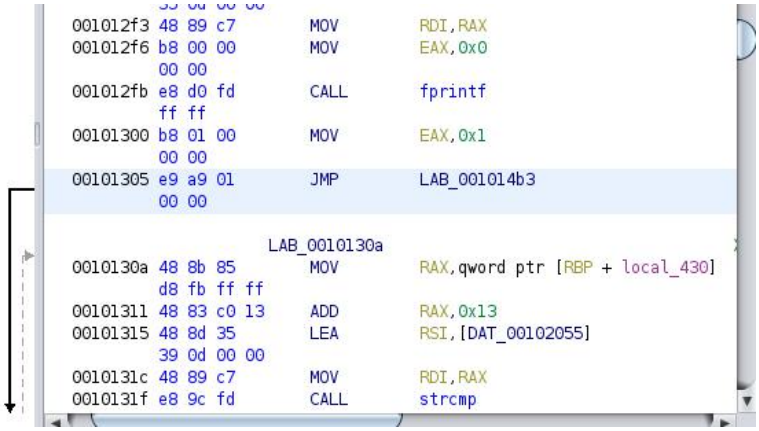
Em seguida, o arquivo usado na encriptação foi baixado através da URL http://ix.io/2c6V e também analisado utilizando o Ghidra. Sua função principal é a “main”. Nela, há diversas verificações a respeito de erros, como de endereço de memória utilizado para acesso do mesmo e de buffer overflow ao final do código. Dessa maneira, no código Assembly, são usados diversos comandos JMP para contornar trechos do código, caso em algum momento erros dessa natureza sejam identificados.




O entendimento do código revela como ocorre a encriptação do diretório do usuário. O código utiliza diferentes variáveis e ponteiros, como pode ser visto abaixo.



A primeira condição analisa o primeiro parâmetro da função (param_1), proveniente da inicialização da função. Se for falsa, há uma nova verficação. Se a função “opendir” retornar um endereço inválido, assim como no caso anterior, é armazenado o valor 1 na variável uVar3 e o ponteiro na pilha de comandos vai para “LAB_001014b3” (sai da estrutura if/else em C, ou seja, utiliza JMP em Assembly), onde analisa-se se ocorreu buffer overflow.

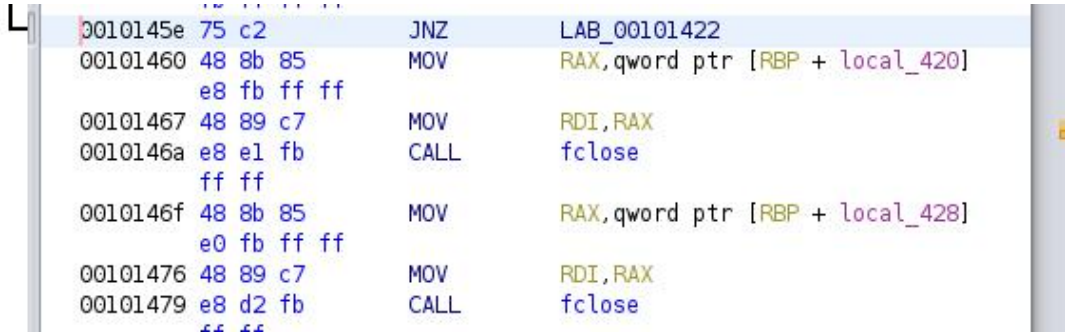



Se o endereço do diretório apontado por um ponteiro pdVar5 não for o último ou se um erro não tiver acontecido, haverá a execução em loop de um trecho de código. Sempre que o ponteiro na pilha chegar ao último comando no bloco “while”, ele retorna para o primeiro endereço que contém o código Assembly daquele trecho.

Após verificar que o nome da pasta é válida, é usado um buffer “local_418”, que armazenará os caminhos originais do diretório do usuário. Após sua criação, essa estrutura é aberta para leitura e escrita, sendo utilizada mais à frente pelo código. Antes de dar prosseguimento, há uma verificação da validade do endereço de memória apontado ao abrir “local_418”. Se ocorrer um erro, o código passa a ser executado a partir de “LAB_001014b3”.

O trecho seguinte do código é responsável pela encriptação do diretório do usuário. Primeiro, é criada uma nova estrutura (“local_218”), na qual serão armazenados os novos caracteres dos

nomes dos componentes do diretório. O novo caracter é o equivalente do inteiro resultante da soma do inteiro correspondente ao caracter original e do inteiro correspondente ao segundo caracter no segundo parâmetro da função “main”. A operação é realizada para todos os caracteres de cada nome de cada pasta para, posteriormente, realizar a mesma coisa para outro nome de pasta do diretório.



```
0010145e 75 c2           JNZ        LAB_00101422
00101460 48 8b 85        MOV        RAX,qword ptr [RBP + local_420]
         e8 fb ff ff
00101467 48 89 c7        MOV        RDI,RAX
0010146a e8 e1 fb        CALL       fclose
         ff ff
0010146f 48 8b 85        MOV        RAX,qword ptr [RBP + local_428]
         e0 fb ff ff
00101476 48 89 c7        MOV        RDI,RAX
00101479 e8 d2 fb        CALL       fclose
```

```
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);
```