Linx Impulse

RELATÓRIO TÉCNICO DO
Processo seletivo para Cloud Engineering

Nome: **Alessandro Elias**

E-mail: ale.elias2011@gmail.com

fone: +55 42 99888 8962

Curitba

02 de fevereiro de 2020

Sumário

1 DESAFIO - Parte 1

Estamos enviando pelos links abaixo alguns binários executáveis (ELF 64-bit LSB) que realizam tarefas bem simples, que podem ou não ser úteis. O exercício é que você descubra o que esses binários fazem, utilizando as ferramentas que julgar mais adequadas. Como resposta, esperamos que você nos diga o que você acha que eles fazem e quais foram as ferramentas usadas para isso, bem como uma linha geral do seu raciocínio para chegar às conclusões.

Nesta seção, para encontrar as soluções dos problemas propostos, foi utilizado primordialmente as seguintes ferramentas:

```
GNU objdump (GNU Binutils) 2.33.1
Copyright (C) 2019 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or (at your option) any later version.
This program has absolutely no warranty.
```

Esta ferramenta foi utilizada para obter o desmontagem (disassembly) dos binários, bem como fazer uma analise das seções e símbolos dos binários. Desta forma, possibilitando uma leitura do código montado (assembly).

Os seguintes comandos foram utilizados para analise das seções e ou cabeçalhos:

Para analises de cabeçalhos e seções

$$objdump\ -x\ <nome\_do\_arquivo>$$

$$objdump\ -T\ <nome\_do\_arquivo>$$

$$objdump\ -f\ <nome\_do\_arquivo>$$

Para disassembly:

$$objdump\ -S\ <nome\_do\_arquivo>$$

$$objdump\ -D\ <nome\_do\_arquivo>$$

$$objdump\ -d\ <nome\_do\_arquivo>$$

```
GNU gdb (GDB) 8.3.1
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

Esta possibilitou a confirmação de qualquer conjectura obtida pela ferramenta anterior, executando o assembly passo a passo.

Sua execução com os seguintes parâmetros:

$$gdb\ -tui\ <programa>$$

Os binários e seus respectivos disassembly, um pequeno leia-me podem ser encontrados no repositório:

https://github.com/alessandro11/desafio-1

1.1 Salvando entrada do usuário em /tmp/<USER>

Binário: cc9621

Disassembly: https://github.com/alessandro11/desafio-1/blob/master/cc9621.as

Este programa abre um arquivo no diretório "/tmp/", sendo nome do arquivo o usuário obtido do ambiente de execução do programa. Espera a entrada do usuário, quando finalizado com um *Enter*, o dado entrado é salvo no arquivo "/tmp/USER".

```
000000000040072d <main>:
  40072d:       push   %rbp
  40072e:       mov    %rsp,%rbp
  400731:       push   %rbx
  400732:       sub    $0x1f8,%rsp
  400739:       mov    %fs:0x28,%rax
  400742:       mov    %rax,-0x18(%rbp)
  400746:       xor    %eax,%eax
  /*
   * The instruction below assign the string "/tmp/".
   * Note the optimization of the compiler result in a
   * direct assignment, no memoey access.
   * (Little endian) 0x2f=/ 0x74=t 0x6d=m 0x70=p 0x2f=/
   * The value "/tmp/" is stored at a variable on the stack.
   */
  400748:       movabs $0x2f706d742f,%rax
  400752:       mov    %rax,-0x1f0(%rbp)
  400759:       lea    -0x1e8(%rbp),%rsi
  400760:       mov    $0x0,%eax
  400765:       mov    $0x18,%edx
  40076a:       mov    %rsi,%rdi
  40076d:       mov    %rdx,%rcx
  /*
   * The following instruction fill the buffer with zeros pointing to
   * register %rsi, by %rcx times (0x18 = 24 chars). In C usually the
   * call would be memset(buff, 24, '\0');
   */
```

```
400770:       rep stos %rax,%es:(%rdi)
/* The address 0x4008b4 point to string "USER". */
400773:       mov    $0x4008b4,%edi
/*
 * The call getenv("USER"); it will return a char* at %rax with the user
 * logged.
 */
400778:       callq  4005b0 <getenv@plt>
40077d:       mov    %rax,-0x200(%rbp)
400784:       mov    -0x200(%rbp),%rdx
40078b:       lea    -0x1f0(%rbp),%rax
400792:       mov    %rdx,%rsi
400795:       mov    %rax,%rdi
/*
 * strcat(%rdi, %rsi) will concatenate the strings:
 * %rdi = "/tmp/" with, %rsi = "m3cool"; results the string
 * "/tmp/m3cool".
 * WARNING:
 *          m3cool is my user at my computer, so this program
 * when run from other computer/user may change.
 * the result string is stored at variable on the stack at offset
 * -0x1f0 from the base pointer register.
 */
400798:       callq  400630 <strcat@plt>
40079d:       lea    -0x120(%rbp),%rax
4007a4:       mov    %rax,%rsi
/* 0x4008b9 stores the string "%s" */
4007a7:       mov    $0x4008b9,%edi
4007ac:       mov    $0x0,%eax
/* scanf waits for a string to be typed, as the parameter at
 * %edi = "%s" indicates
 */
4007b1:       callq  400620 <__isoc99_scanf@plt>
4007b6:       lea    -0x1f0(%rbp),%rax
/* 0x4008bc stores string "w" for fopen */
4007bd:       mov    $0x4008bc,%esi
4007c2:       mov    %rax,%rdi
/*
 * tries to open a file for write only.
 * -0x1f8(%rbp) = fopen("/tmp/m3cool", "w");
 */
4007c5:       callq  400610 <fopen@plt>
4007ca:       mov    %rax,-0x1f8(%rbp)
4007d1:       cmpq   $0x0,-0x1f8(%rbp)
4007d9:       je     40080a <main+0xdd>
4007db:       mov    -0x1f8(%rbp),%rdx
4007e2:       lea    -0x120(%rbp),%rax
```

```
4007e9:       mov    %rdx,%rsi
4007ec:       mov    %rax,%rdi
/*
 * In case fopen succeeded, saves the content typed
 * in the file opened at "/tmp/m3cool".
 */
4007ef:       callq  4005e0 <fputs@plt>
4007f4:       mov    -0x1f8(%rbp),%rax
4007fb:       mov    %rax,%rdi
/* closes the stream file opened at "/tmp/m3cool"
4007fe:       callq  4005c0 <fclose@plt>
/*
 * The remaining instructions is to return and
 * shutdown the process gracefully.
 */
400803:       mov    $0x0,%eax
400808:       jmp    40080f <main+0xe2>
40080a:       mov    $0x1,%eax
40080f:       mov    -0x18(%rbp),%rbx
400813:       xor    %fs:0x28,%rbx
40081c:       je     400823 <main+0xf6>
40081e:       callq  4005d0 <__stack_chk_fail@plt>
400823:       add    $0x1f8,%rsp
40082a:       pop    %rbx
40082b:       pop    %rbp
40082c:       retq
40082d:       nopl   (%rax)
```

Assembly 1: Escreve entrada do usuário em "/tmp/USER".

No Assembly 1 é mostrado o *main* do programa, as outras seções foram truncadas, assim focamos apenas no núcleo de que o programa faz (não irei abordar cada instrução, apenas as relevantes). Também é apresentado como comentários em inglês mais detalhes das instruções relevantes.

A primeira instrução relevante é 4007{48,52}, após a linha 15. No qual armazena a string "/tmp/" em uma variável na pilha. A instrução 400770 preenche com zeros um buffer de tamanho 24, indicado pela constante armazenado em *RCX*, no qual controla quantas vezes será repetido a instrução *stos*. Este buffer será usado pelo (scanf). Na instrução 400778 é obtido o usuário que está executando o programa, em meu caso o usuário é "m3cool". A 400798 concatena a string "/tmp/" com "m3cool". Na 4007b1 é feito a chamada ao *scanf* para obter a entrada do usuário, uma string qualquer. A 4007c5 abre o arquivo no caminho "/tmp/m3cool". Na 4007ef ((fputs) escreve a string obtida pelo *scanf* no arquivo aberto. Na 4007fe fecha o arquivo e o programa é encerrado.

1.2 Conway's Game of Life

Binário: d3ea79

Disassembly: https://github.com/alessandro11/desafio-1/blob/master/d3ea79.as

Este programa gera a saída do jogo Conway's Game of Life, escrevendo-o no arquivo "/tmp/<USER>". O link [1] para a Wiki do jogo também é escrito no arquivo. A saída gerada na execução deste programa está no repositório em /tmp/m3cool.

O disassembly parcial do binário pode ser observado nas rotinas abaixo:

```
00000000004011c7 <main>:
...
  4011eb:       callq  40094d <clear>
...
  4011f5:       callq  4009e6 <welcome>
...
  401208:       callq  400fc2 <play>
```

Assembly 2: Remove e recria o arquivo "/tmp/<USER>" com o Conway's Game of Life

No Assembly 2 observamos as respectivas chamadas: *clear*, *welcome play*. Iremos abordar cada uma das rotinas.

clear:

```
000000000040094d <clear>:
...
  400967:       movabs $0x2f706d742f,%rax
...
  400992:       mov    $0x4012a8,%edi
  400997:       callq  400750 <getenv@plt>
...
  4009bc:       callq  4007b0 <strncat@plt>
...
  4009cb:       callq  400760 <unlink@plt>
...
```

Assembly 3: Remove arquivo "/tmp/<USER>"

A instrução 400967 atribui ao registrador *RAX* a string "/tmp/". Na instrução 400992 é atribuído o ponteiro para a string (constante) "USER", no qual é passado como parâmetro para *getenv*. Este retorna o valor da variável de ambiente <USER>. A instrução 4009bc concatena a string que estava em *RAX* e valor da variável de ambiente. No meu caso, resultando na string "/tmp/m3cool". A 4009cb remove o arquivo "/tmp/m3cool".

welcome:

[1] http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

```
00000000004009e6 <welcome>:
...
  400a00:       movabs $0x2f706d742f,%rax
...
  400a2b:       mov    $0x4012a8,%edi
  400a30:       callq  400750 <getenv@plt>
...
  400a55:       callq  4007b0 <strncat@plt>
...
  400a69:       callq  400810 <fopen@plt>
...
  400aa7:       mov    $0x4012c3,%edi
  400aac:       callq  400830 <fwrite@plt>
...
  400ab8:       mov    $0x4012e8,%esi
  400ac5:       callq  4007f0 <fprintf@plt>
...
  400ad4:       callq  400780 <fclose@plt>
...
```

Assembly 4: Escreve cabeçalho no arquivo "/tmp/<USER>"

O Assembly 4, na instrução 400a00 à 400a55 concatena strings como em 3. Instrução 400a69 abre o arquivo ("/tmp/m3cool"). Na instrução 400aa7 carrega ponteiro para string (constante) "Welcome to the Game of Life.\n" a subsequente escreve no arquivo esta string. A instrução 400ab8 carrega a string (constante) "http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life" e subsequente escreve no arquivo e então fecha-o.

play, ***print***, ***evolve*** são rotinas responsáveis por gerar o jogo.

1.3 Fork()

Binário: da87fa

Disassembly: https://github.com/alessandro11/desafio-1/blob/master/d3ea79.as

Este programa fica em um laço infinito, ou até receber um SIGHUP, executando *fork()* dez vezes. Na décima, é executado um *sleep* de dez segundos, e torna a executar outros dez forks. Como é mapeado SIGCHLD para SIGHUP, o processo filho inicia e encerra.

Analisando a desmontagem do binário no Assembly 5:

```
000000000040060d <main>:
  40060d:       push   %rbp
  40060e:       mov    %rsp,%rbp
  400611:       push   %rbx
  400612:       sub    $0x38,%rsp
```

```
400616:     mov     $0x1,%esi
40061b:     mov     $0x11,%edi
400620:     callq   4004d0 <signal@plt>
400625:     mov     $0x0,%ebx
40062a:     jmp     400650 <main+0x43>
40062c:     callq   400510 <fork@plt>
400631:     movslq  %ebx,%rdx
400634:     mov     %eax,-0x40(%rbp,%rdx,4)
400638:     movslq  %ebx,%rax
40063b:     mov     -0x40(%rbp,%rax,4),%eax
40063f:     test    %eax,%eax
400641:     jne     40064d <main+0x40>
400643:     mov     $0x0,%edi
400648:     callq   4004f0 <exit@plt>
40064d:     add     $0x1,%ebx
400650:     cmp     $0x9,%ebx
400653:     jle     40062c <main+0x1f>
400655:     mov     $0xa,%edi
40065a:     mov     $0x0,%eax
40065f:     callq   400500 <sleep@plt>
400664:     jmp     400625 <main+0x18>
400666:     nopw    %cs:0x0(%rax,%rax,1)
```

Assembly 5: Fork do processo

As instruções 4006{16, 1b, 20} mapeiam o sinal de SIGCHLD para SIGHUP, o que acarreta na finalização do processo, assim que o processo filho inicia. Da instrução 40062a à 400653 ocorre o laço dos forks. Nas instruções 40065{0, 3} controla quantos forks serão feitos enquanto o registrador *EBX* não atinge dez, um if. Quando o registrador *EBX* atinge dez, então é executado um *sleep* de dez segundos e volta para o inicio do laço.

O interessante para sair do laço, finalizando o processo, nunca ocorrerá. Conforme as instruções 40063f executa uma comparação *test*, no qual somente será igual se o registrador *EAX* for zero, porém a instrução lê o pid do processo filho que foi salvo na pilha, atribui este valor ao registrador *EAX* e executa o *test*, logo não é atribuído pid zero para processos, por definição do SO. Então a instrução 400641 sempre fará o salto para a verificação se já executou os dez forks antes do *sleep*, e nunca executará 400648, finalizar processo.

1.4 Multi Thread escutando na porta 8011

Binário: ddb1c9

Disassembly: https://github.com/alessandro11/desafio-1/blob/master/ddb1c9.as

Solução ainda parcial e não conclusiva.

2 DESAFIO - Parte 2

O ambiente deve ser todo configurado através de gerenciador de configuração, o que deverá ser entregue é um repositório git contendo os arquivos de configuração que serão aplicados em uma máquina virtual "zerada". Caso necessário descrever como executar o processo de aplicação da configuração na máquina virtual. Ao final da tarefa e execução do processo, deveremos ter um ambiente funcional;

- É recomendado que o repositório git seja entregue com commits parciais, mostrando a evolução de seu código e pensamento. Caso prefira nos informe um url de git público ou então compacte todos os arquivos em um .tar.gz mantendo a pasta .git em conjunto;

- No ambiente deverá estar rodando uma aplicação node.js de exemplo, conforme código abaixo. A versão do node.js deverá ser a última versão LTS disponível em: https://nodejs.org/en/download/. A aplicação node abaixo possui a dependência da biblioteca express. Garanta que seu processo de bootstrap instale essa dependência (última versão estável disponível em: http://expressjs.com/) e rode o processo node em background. De uma forma dinâmica garanta que seja criado uma instância node para cada processador existente na máquina (a máquina poderá ter de 1 a 32 processadores);

Construa dentro de sua automação um processo de deploy e rollback seguro e rápido da aplicação node. O deploy e rollback deverá garantir a instalação das dependências node novas (caso sejam adicionadas ou alteradas a versão de algum dependência por exemplo), deverá salvar a versão antiga para possível rollback e reiniciar todos processos node sem afetar a disponibilidade global da aplicação na máquina;

- A aplicação Node deverá ser acessado através de um Servidor Web configurado como Proxy Reverso e que deverá intermediar as conexões HTTP e HTTPS com os clientes e processos node. Um algoritmo de balanceamento deve ser configurado para distribuir entre os N processos node a carga recebida;

- A fim de garantir a disponibilidade do serviço, deverá estar funcional uma monitoração do processo Node e Web para caso de falha, o mesmo deve reiniciar ou subir novamente os serviços em caso de anomalia;

- Desenvolva um pequeno script que rode um teste de carga e demostre qual o Throughput máximo que seu servidor consegue atingir;

- Desenvolva um script que parseie o log de acesso do servidor Web e deverá rodar diariamente e enviar por e-mail um simples relatório, com a frequência das requisições e o respectivo código de resposta (ex:5 /index.html 200);

- Por fim; rode o seu parser de log para os logs gerados pelo teste de carga, garantindo que seu script terá performance mesmo em casos de logs com milhares de acessos;