Relatório Trabalho 1

Victor Ribeiro Garcia
Departamento de Informática
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
vrg20@inf.ufpr.br

***Resumo* – O seguinte relatório do Trabalho 1 da disciplina Arquitetura de Computadores (CI1212) apresenta de forma sucinta a ISA do processador x86-64 e diferencia do conjunto de instruções do MIPS32. Também, será demonstrado de que maneira as senhas de acesso para um arquivo binário foram encontradas e como os desafios propostos (modificação do binário para remoção da solicitação de senha e um gerador de senhas válidas para o programa) foram implementados.**

I. Introdução

A ISA (*Instruction Set Architecture*) é parte do modelo abstrato de um computador que define como a CPU é controlada pelo software. Inicialmente, a arquitetura x86 aparece como o Intel 8086 CPU (baseado em 16 bits) lançado em 1978. Neste relatório foi abordada a ISA presente nos processadores de arquitetura x86-64 (ampliação do x86 para 64 bits feita pela AMD em 2003). Adiante, este conjunto de instruções será explicado e comparado com o dos processadores MIPS32 baseados em 32 bits.

Além disso, foram disponibilizados arquivos binários para cada aluno com senhas de acesso distintas. Neste caso, o arquivo referente a esse relatório foi o “prog2” ao qual foi compilado em uma máquina x86-64 e serão demonstrados métodos utilizados para encontrar as senhas e suas peculiaridades.

II. Comparando as ISAs do x86-64 e MIPS32

As arquiteturas x86-64 e MIPS32 seguem *General Register based CPU Organization*, isso significa que são consideradas arquiteturas com registradores de uso geral (podem ser usados para endereços ou para dados). O x86-64 apresenta 16 registradores de uso geral e podem ser acessados das seguintes formas: 8 (byte), 16 (palavra - *word*), 32 (palavras duplas - *doubleword*), 64 (quatro palavras - *quadword*) e 128 (*double quadword*) bits de extensão; o MIPS32 possui 32 registradores e todas instruções ocupam 32 bits.

No x86 as instruções aritméticas, lógicas e de transferência de dados são instruções de dois operandos (precisam ter um dos operandos atuando como origem e destino), já o MIPS tolera apenas registradores separados para origem e destino. Essa limitação coloca mais pressão sobre os registradores no x86, pois um registrador de origem precisa ser modificado. Outra diferença é que um dos operandos pode estar na memória, diferentemente do MIPS (no x86, uma instrução aritmética pode ser feita de um valor na memória com um valor imediato; no MIPS32, primeiro o valor teria que ser carregado na memória por uma instrução e, depois, outra faria a operação aritmética).

Nas *branches* (instrução de desvio condicional) do x86-64 existe uma peculiaridade em relação ao MIPS32. Nessas instruções, as flags são armazenadas em um registrador de código condicional.

III. Instruções do x86-64

As instruções do x86-64 podem ter entre 1 e 15 bytes de comprimento. O comprimento é definido de acordo com cada instrução, dependendo dos modos de operação disponíveis da instrução. O formato geral de uma instrução do x86 é constituído por: *Prefix* (modifica o comportamento da instrução), *Opcode* (código da operação), *ModR/M* (especifica os locais dos operandos e como eles são acessados), *SIB* (localiza e define o modo de acesso dos operandos) e *Data* (imediato).

A seguir são apresentadas algumas instruções do x86-64 para exemplificar o funcionamento:

- Instruções de movimentação de dados:
  mov rbx, 5 (rbx = 5)
- Instruções aritméticas e lógicas:
  add reg, r/m (reg = reg + r/m)
- Instruções de fluxo de controle:
  cmp r/m, reg (compara r/m com reg e define RFLAGS de acordo)

Nos exemplos acima, foi utilizada a notação da Intel: instrução destino, fonte; a sintaxe AT&T segue um modelo diferente: instrução fonte, destino.

IV. Senhas do arquivo prog2

O arquivo prog2 é um dos programas do *coreutils* do GNU modificado de forma a pedir uma chave. Ao executar o programa, ocorre o pedido de uma senha e caso a mesma seja inválida, uma mensagem de senha incorreta é impressa:

Para acessar o programa, digite sua chave: 100
Chave invalida. Terminando o programa

Primeiramente, a ideia foi realizar o *diassemble* (desmontar) o binário utilizando o comando *objdump* com a opção -d (*diassemble).* Após isso, foi utilizada a ferramenta *gdb* (depurador do GNU) com o comando *info functions* no arquivo para listar todas declarações de funções e encontrou-se a seguinte função:
57: char validarSenha(unsigned long, unsigned long);
Em seguida, para buscar mais informações da função, foi utilizado o comando do gdb nomeado *break (*pausará o programa quando a função é chamada).

(gdb) break validarSenha
Breakpoint 1 at 0x2657: validarSenha. (3 locations)
(gdb) run
...
Para acessar o programa, digite sua chave: 100
Breakpoint 1, validarSenha (senha=100, val=1362) at src/whoami.c:57

O primeiro BreakPoint foi apontado em 0x2657. Ao analisar essa localização na saída do comando objdump, é encontrada dentro da main.

```
264e:  callq   2570      <SCANF>
2653:   mov   (%rsp),%rax
2657:  lea    -0x552(%rax),%rdx
265e:  cmp   $0x17,%rdx
2662:  jbe    2677 <main+0x67>
...
2677:  test   $0x1,%al
2679:  jne    2664 <main+0x54>
...
```

Após a leitura (*scanf*) da senha, esse trecho da main verificará se o valor digitado é válido. A senha é subtraída de 1362 (-0x552) e o resultado é comparado usando a instrução *jbe* (saltar se for inferior ou igual) com o valor 23 (0x17). Assim, o intervalo de senha está entre 1362 e 1385 (1362+23). Porém, o salto feito por *jbe* vai para a instrução *test* que está verificando se o valor é par. Portanto, as senhas são valores pares entre 1362 e 1385.

Para acessar o programa, digite sua chave: 1374
Chave correta, voce pode usar o programa.
vrg20

Como visto acima, a quebra de senha de um programa que não apresenta métodos de segurança não é complexa (pode-se utilizar de métodos como engenharia reversa ou força bruta). Por isso, deveria ter sido aplicado algum procedimento de criptografia como Scrypt, Bcrypt, entre outros.

V. Desafios

I. Desafio 1

O primeiro desafio proposto tratava-se de uma modificação no binário prog2 com o objetivo de que ele não solicite mais senhas. Para isso, poderia ter sido utilizado o comando *nop* ou *jmp.*

```
000000000002610 <main>:
   ....
2636:      48 89 44 24 08      mov    %rax,0x8(%rsp)
263b:      31 c0               xor    %eax,%eax
263d:      e8 ee fe ff ff      callq  2530 <__printf_chk@plt>
2642:      48 89 e6            mov    %rsp,%rsi
```

A instrução *xor* (31 c0) está antes do *printf("Para acessar o programa, digite sua chave: ")* e possui a quantidade de bits semelhante a de um *jmp* (dois bytes). Assim, essa instrução deveria "pular" a parte do código onde ocorre a verificação da senha totalizando 62 bytes (3E). Abaixo está um trecho do código

apresentando a modificação do código feito no *hexedit* (editor hexadecimal) e para onde a instrução foi após o salto.

000000000002610 <main>:

...

263b: eb 3e jmp 267b <main+0x6b>

...

267b: 48 8d 3d 76 3c 00 00 lea 0x3c76(%rip),%rdi

Assim, a saída após a edição do código ficara dessa forma:

Chave correta, voce pode usar o programa.
vrg20

II. *Desafio 2*

Este desafio consistia na criação de um *Key Generator* (programa na linguagem C que gere senhas válidas aleatórias para o arquivo prog2). Na construção do código foi utilizada a função *aleat* para gerar números aleatórios em um intervalo (no caso de prog2 foram senhas entre 1362 a 1385) e uma verificação para números pares (senha % 2 != 0).

VI. Conclusão

Nesse relatório foram feitas abordagens em linguagem de programação de baixo nível (x86-64 e MIPS32). Assim, decorreu-se para entender o seu funcionamento e compreender um programa em hexadecimal sem mesmo possuir o código fonte.

Referências

- D. Patterson; J. Henessy. Organização e Projeto de Computadores: a Interface Hardware/Software. 5a Edição. Elsevier Brasil, 2017.
- STALLINGS, W. Arquitetura e Organização de Computadores. 10 ed. Prentice Hall. São Paulo, 2018.
- Bob Plantz. Introduction to Computer Organization: A Guide to X86-64 Assembly Language and GNU/Linux. 2011.
- William Mahoney; J. Todd McDonald. Enumerating x86-64 – It's Not as Easy as Counting. Acesso em: 04 Jul. 2022. URL: https://www.unomaha.edu/college-of-information-science-and-technology/research-labs/_files/enumerating-x86-64-instructions.pdf