Engenharia de Segurança

Aula 10 - 23/04/2018

Afonso Fontes (pg35389)

Bruno Carvalho (a67847)

Mariana Carvalho (a67635)

30 de Abril de 2018

Universidade do Minho

1 - Buffer Overflow

Pergunta 1.1

No caso dos programas escritos em **Java** e **Python** respetivamente, quando existe uma tentativa de aceder a uma posição de memória não válida, ou seja, a uma posição do *array* buffer que ultrapasse os limites do próprio é disparada uma exceção ,sendo encerrada a respetiva *thread*, isto acontece porque ao contrário do que acontece na mesma definição escrita em **C++** , aqui a noção de *array* prevalece em *runtime*, sendo verificada cada tentativa de acesso ao mesmo, o que não acontece tanto no caso de **C** ou **C++** onde a este nível não existe um controlo correto em caso de *buffer overflow*.

Pergunta 1.2

Neste caso, o comportamento é semelhante ao que foi retratado na **Pergunta 1.1**, quando em tempo de execução é requisitado pelo programa *input* de forma a definir quantos valores se pretende guardar no *array*, caso este valor ultrapasse a dimensão em bytes do array, vai existir uma tentativa de acesso e escrita fora dos limites definidos como memória em stack para o respetivo array "vals", nesta situação quando o programa é executado na sua definição em python ou java, uma exceção não prevista é largada não sendo tratada em nenhuma parte da sua definição sendo por isso, finalizado o programa.

Pergunta 1.3

No primeira definição *RootExploit.c*, de forma a obter acesso sem introdução da frase correta, basta inserir 5 bytes aleatórios, por exemplo "11111", desta forma os quatro primeiros são escritos no array buff, enquanto que o quinto é escrito sobre o espaço de endereçamento reservado para a variável locar *pass*, neste momento *pass* passa a ser diferente de zero, sendo assim verdadeira a condição *if (pass)* , fornecendo acesso sem a password correta ter sido fornecida ao programa. No segundo caso, a especificação em *C* ,*0-simple.c*, de forma a ser possível imprimir a string "YOU WIN", foi para isso nescessário arranjar forma alterar a variável control para um valor diferente de 0. Após análise do programa através do **gdb**, verificamos que o *array* buffer se encontrava a 76 bytes do endereço reservado para a variável *control*, assim, sendo que o input escrito é ele depositado diretamente no array sem qualquer verificação bastou, para realização do exercicio efetuar a inserção de 77 bytes , sendo o último diferente de 0.

Pergunta 1.4

O programa ReadOverflow lê uma string do stdin e guarda-a num array de 100 bytes, imprimindo para stdout o número de caracteres guardados no buffer que o utilizador define. Como se pode observar na imagem abaixo, quando se tenta ler mais do que 100 bytes, o tamanho do buffer, o programa acede e imprime o conteúdo de células de memória fora dos limites do array.

```
user@CSI:codigofonte$ gcc -o ReadOverflow ReadOverflow.c
user@CSI:codigofonte$ ./ReadOverflow
Insira numero de caracteres: 125
Insira frase: seguranca
ECO: |seguranca.............@@@.....@.......@5@L@...@5@L@...@@5@.............@9p1
JV..................@9p1JV..P7p1}...`5@L@.......||
Insira numero de caracteres:
```

Pergunta 1.5

Para colocar o valor pretendido na variável "control", o primeiro passo foi utilizar o gdb para observar os endereços de memória das variáveis "control"e "buffer". Observamos que o endereço de "control"era sempre 76 bytes mais alto que o endereço de "buffer"(e não os 64 bytes que seriam de esperar pela observação do código fonte). Então, para colocar a variável "control"com o valor 0x61626364, é necessário passar como argumento ao programa uma string de 76 caracteres arbitrários, seguidos de "dcba"(pois 0x61 é o código ASCII de 'a', 0x61 de 'b' e assim sucessivamente). A ordem dos caracteres necessita de ser invertida (isto é, passamos primeiro o 0x64 - d e por último o 0x61 - a) porque a arquitectura em que estamos a trabalhar (x86_64) é little endian (os números são representados com os dígitos menos significativos primeiro, ao contrário da notação que normalmente usamos para escrever números).

```
user@CSI:~/1718-EngSeg/TPraticas/Aula10/codigofonte$ ./a.out 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111dcba
You win this game if you can change variable control to the value 0x61626364'
Congratulations, you win!!! You correctly got the variable to the right value
user@CSI:~/1718-EngSeg/TPraticas/Aula10/codigofonte$
```

Pergunta 1.6

Depois de analisado o código e o funcionamento do programa, percebe-se que o objetivo é alterar o valor da variável fp de forma a que esta contenha o valor do endereço da função **win**. Para explorar a vulnerabilidade, o código foi inicialmente compilado com a flag de debug através da instrução **gcc 2-functions.c -g -o 2-functions**, sendo o executável seguidamente analisado com recurso ao debugger gdb, com o comando **gdb 2-functions**. Com o debugger, verificou-se que a função **win** se encontra no endereço 0x555555554740, através do comando **p win**.

```
gdb-peda$ p win
$1 = {void ()} 0x555555554740 <win>
```

Sabido este endereço, converteu-se o valor para código ASCII, o que resultou no código UUUUG@ (@GUUUU quando convertido para little-endian, o modo utilizado nesta arquitetura). Para obter a mensagem pretendida, foram concatenados 72 bytes aleatórios (tendo em conta que é a partir do 72º byte que a variável fp é alterada) com o código obtido.

```
gdb-peda$ run
Starting program: /home/user/1718-EngSeg/TPraticas/Aula10/codigofonte/2-functions
You win this game if you are able to call the function win.'
012345678901234567890123456789012345678901234567890123456789012345678901@GUUUU
calling function pointer, jumping to 0x555555554740
Congratulations, you win!!! You successfully changed the code flow
```

Pergunta 1.7

Para resolver este exercicio procedendo de forma análoga à que foi realizada durante a experiência 1.7, utilizando o *debugger*, após criar um *breakpoint* aquando da chamada da função *gets*, identificamos a posição do endereço de retorno, assim como o endereço da função "win"que pretendemos executar, e por fim o endereço do buffer onde seria possível efetuar a escrita. Executando um processo semelhante aquele apresentado durante a experiência 1.7 e à respetiva aula teórica, introduzimos os bytes necessários para a partir do endereço inicial do buffer, atingir o endereço do **endereço de retorno (rip)**, finalizando com a escrita, neste mesmo endereço dos bytes que identificam a posição da função **win**. Após várias tentativas não nos foi possível concluir o exercício com sucesso devido a um *segmentation fault* antes do print ter sido executado.