Universidade do Minho

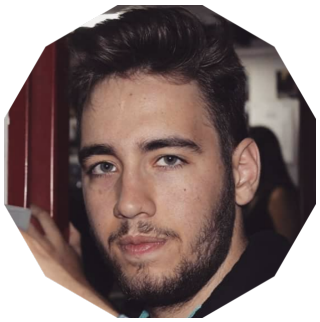
Mestrado Integrado em Engenharia Informática

Engenharia de Segurança

Aula 9 - Buffer Overflow

Grupo 8

João de Macedo
A76268



João Aloísio
A77953

Nelson Gonçalves
A78173

27 de Abril de 2020

1 Pergunta 1.1

Este mesmo programa escrito nas três linguagens, C++, Python e Java, tem comportamentos diferentes. Neste programa é pedido a alocação de um pedaço de memória estática para que seja possível armazenar 10 inteiros. Posto isto, é pedido ao utilizador para dizer quantos inteiros pretende inserir, sendo que insere os inteiros pretendidos, sabendo o número máximo.

Sabendo isto, como o espaço alocado é estático, ao tentar armazenar um maior número de inteiros, os programas têm comportamentos diferentes, sendo que no caso da implementação em C++ irá resultar "stack smashing", em Java, a execução será interrompida com uma exceção : **Exception in thread "main"java.lang.ArrayIndexOutOfBoundsException: 10 at LOverflow2.main(LOverflow2.java:18)** e por último em Python, tal como em Java é lançada uma exceção **IndexError: list assignment index out of range**.

2 Pergunta 1.2

A vulnerabilidade de *Buffer Overflow* existente nestes programas denomina-se por **Strack-based buffer overflow**. Quando um *array* é declarado em C, o espaço para ele é reservado e este array é manipulado pelo seu apontador, para o primeiro byte. Tendo isto em conta, a função *gets* não é segura pois esta não verifica os limites do *array* podem ser ultrapassados, ou seja, o programa permite que copie informação, para além dos limites do *array*, alterando assim o conteúdo das posições de memória adjacentes.

No programa **RootExploit** se for usado uma string com um comprimento de mais de 4 caracteres, pode-se alterar a informação nas posições de memória adjacentes, alterando assim o valor da variável **pass** e obter permissões de root/admin.

Já no programa **0-simple**, se for introduzido uma string com um comprimento maior que 64, é possível alterar o valor da variável *control* e obter a mensagem "YOU WIN!!!".

3 Pergunta 1.3

Após testar e executar o programa 'ReadOverflow.c', podemos concluir que, facilmente, se extrai conhecimento remanescente da memória, executando este programa de forma a que o mesmo seja quebrado. Um exemplo disto é indicar ao programa que pretendemos escrever 20 caratéres, mas quando escrevermos a frase, escrevemos uma que tenha apenas 9 caratéres, por exemplo. O programa ao percorrer o ciclo sem validar se o tamanho da frase é, de facto, igual ao tamanho inicialmente indicado, para além de imprimir os caratéres inseridos, também vai imprimir os restantes que faltam até atingir o número indicado, de posições sucessivas da memória, que ficaram como dados remanescentes.

4 Pergunta 1.5

Para mitigar as vulnerabilidades do *Buffer Overflow* foram usadas as técnicas apresentadas na aula teórica ,tais como:

- **Espaço alocado:** Para implementar esta técnica foi utilizada a função *malloc_usable_size()* que nos permite saber o espaço alocado, em que pelo menos nos dá o mínimo pedido, sendo que este nunca é exato. Com isto foi possível saber se o tamanho do argumento dado era superior ao tamanho do espaço alocado em memória.
- **Evitar utilizar funções de risco:** Foi alterado o *strcpy* para *strncpy* sabendo que esta é mais segura pois é dado à função o tamanho disponível para o buffer destino. O grupo tentou usar o *strlcpy* sabendo que este é o mais seguro mas não conseguiu usar a biblioteca necessária para tal.

5 Pergunta 1.6

As alterações efectuadas no programa stack.c foram as seguintes

- Na função *buf* :
  - Além de receber a string como argumento também recebe o tamanho dela. De seguida é feita a confirmação de que o tamanho recebido como argumento corresponde de facto ao tamanho da string, caso não seja é retornado 0.
  - O é alocada memória do buffer criado consoante o tamanho da string
- Na função main:
  - A string str é declarada sem tamanho inicial.
  - É calculado o tamanho de bytes do badfile, e é alocada memoria à string consoante esse tamanho.
  - Antes de escrever o que foi lido da file, é verificado se está leu alguma coisa ou nao, caso dê null é definido o tamanha da string que retorna como 0
  - Um dos parâmetros do fread é o tamanho do ficheiro *fread(str,sizeof(char),tam_badfile,badfile)*
  - Os parâmetros passado á função bof é a str e a len dela