# Introdução

TEXTO AQUI

# Parte 1- *ShellShock*

Nesta primeira parte do trabalho, será explorada uma vulnerabilidade na aplicação *bash* usada para executar comandos nos terminais de sistemas baseados em *Linux*, a qual ficou conhecida como *ShellShock*.

Uma das maneiras de explorar esta vulnerabilidade era através de pedidos *HTTP*, que será usada nesta parte, mas havia outras maneiras como *SSH* ou *DHCP*.

## Inicialização do ambiente

Para estudar esta vulnerabilidade, o docente disponibilizou uma *Docker-Image* que contém a versão vulnerável do *bash*. Após inicializar um contentor com a imagem vulnerável, seguindo o enunciado do trabalho, foi utilizado o *browser* para verificar a sua inicialização.

A screenshot of a computer screen

AI-generated content may be incorrect.

Figura 1 - /cgi-bin/getenv.cgi

A screenshot of a computer

AI-generated content may be incorrect.

Figura 2 - /cgi bin/vul.cgi

Na figura 1 e 2, podemos ver que ambos os programas estão inicializados e a responder com o *output* esperado segundo o enunciado do docente e o enunciado do *SEED Labs*.

## Vulnerabilidade

Segundo o enunciado do *SEED Labs*, capítulo “*3.2 Task 2: Passing Data to Bash via Environment Variable*” para explorar esta vulnerabilidade do programa *bash* o atacante precisa de passar a sua informação ao programa *bash* através de variáveis de ambiente.

Quando um pedido *HTTP* é enviado ao servidor, através do *browser* ou através da ferramenta *Curl*, o servidor guarda certas informações presentes no *header* do pedido em variáveis de ambiente.

Para manipular o conteúdo enviado no *header* do pedido *HTTP* foi usada a ferramenta *Curl*. Através desta ferramenta descobrimos que se passarmos informação no *header* usando as *flags*:

* -A : modifica o parâmetro *User\_Agent* do *header;*
* -e : modifica o parâmetro *Refer* no *header;*

A informação passada usando estas *flags* é guardada na variável de ambiente *HTTP\_USER\_AGENT*.

## Explorar a vulnerabilidade

Com esta informação, como é que executamos comandos no servidor e, por exemplo, obtemos o conteúdo do ficheiro /etc/passwd? (alínea 2a).

Para obter a informação temos de fazer com que a variável de ambiente seja interpretada como uma função, ou seja, o conteúdo do *header* do pedido *HTTP* deve conter o seguinte formato: **() { :; }; /bin/bash -c “COMANDO”**.

**A computer screen with white text

AI-generated content may be incorrect.**

Figura 3 - Output da execução de comandos no servidor alvo

Na figura 3, é possível observar o uso da *flag* -A e do formato do *header* indicado anteriormente, para apresentar o conteúdo do ficheiro /etc/passwd e para listar a pasta em que o programa se encontra. A *flag* -e produz o mesmo resultado.

A screen shot of a computer

AI-generated content may be incorrect.

Figura 4 - Execução do comando touch

Seguindo o enunciado do docente (alínea 2b), experimentou-se então criar um ficheiro na pasta *tmp* do servidor, enviando a informação apresentada na figura 4.

A computer screen shot of a computer code

AI-generated content may be incorrect.

Figura 5 – Novo ficheiro criado dentro do servidor

Na figura 5, podemos confirmar, dentro do contentor, que o novo ficheiro “NOVAPASTAAQUI” foi criado na pasta *tmp* através do pedido efetuado ao servidor.

Continuando a seguir o enunciado (alínea 2c), é perguntado se é possível obter o conteúdo do ficheiro /etc/shadow do servidor. Após testes, concluiu-se que não é possível, porque este ficheiro só pode ser por utilizadores com permissões adequadas, neste caso, o utilizador *root*. Esta vulnerabilidade permite a execução de comandos, mas não anula a necessidade de permissões para ler ficheiros protegidos.

Por fim (alínea 2d), é questionado se é possível explorar esta vulnerabilidade através da variável de ambiente *QUERY\_STRING*, que é usada para guardar os dados enviados num pedido HTTP GET no URL após a marca ‘?’.

poderia ser usado para a mesma vulnerabilidade porque o resultado da query é guardado na variavel de ambiente QUERY\_STRING, mas a maneira como o browser codifica a string e a maneira como o server a descodifica porque pode modificar os comandos enviados no na string fazendo com que os comandos não sejam executados.

# Parte 2 – CodeQL e ZAP Proxy