Escola de Ciências e Tecnologia
Departamento de Informática
Licenciatura em Engenharia Informática
Unidade Curricular Segurança Informática
Ano letivo 2020/2021

Stack Overflow

Docente:
Professor Pedro Patinho
Discentes:
José Santos - 43017
Filipe Alfaiate - 43315

July 15, 2021

Contents

1 Introdução

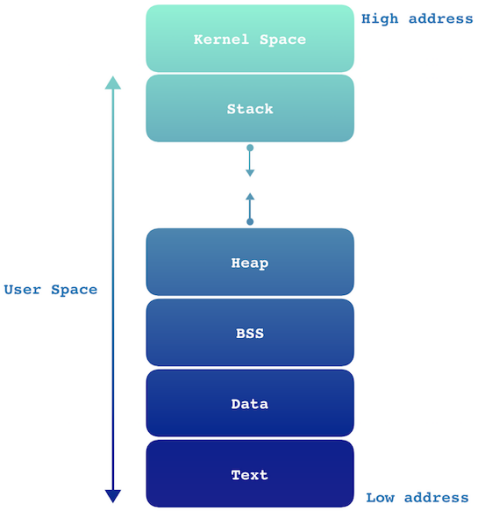
No âmbito da unidade curricular de **Segurança Informática**, lecionada pelo professor **Pedro Patinho**, foi solicitado um trabalho escrito com o tema escolhido pelos alunos e um trabalho prático com a implementação desse mesmo tema.

Face a isto, o tema escolhido foi o `Stack Overflow`. Este refere-se especificamente ao caso em que a `stack` cresce para além do espaço reservado da mesma. Um exemplo prático para que isso aconteça é quando uma chamada de uma função recursiva não termina, o que irá gerar `Stack Overflow`, pois cada chamada cria uma nova `Stack Frame` e a `stack`, eventualmente, alcançará uma zona de memória que não devia.

2 Exploit

Antes de explicarmos o que é um `Stack Overflow`, importa definir o que é um `exploit`. Este pode ser um programa, uma porção de dados ou uma sequência de comandos que se aproveita das vulnerabilidades de um `Sistema Operativo` (SO). É utilizado com a finalidade de aceder ao programa vulnerável executando operações que não eram suposto ocorrerem.

3 Diferentes tipos de ataques na memória



3.1 Heap Overflow

Para melhor entendermos como funciona um `Heap Overflow` é fundamental compreender o que é, e como funciona o `heap`. A `heap` é uma região de memória reservado pelo SO, onde cada processo armazena os dados referentes às variáveis, espaço este que é alocado dinamicamente durante a sua execução.

`Heap Overflow`, também conhecido como `Heap Overrun`, é uma anomalia num programa no qual, ao se gravar os dados de uma região `heap`, ocorre a superação do limite da reserva e a substituição da memória adjacente.

3.2 Buffer Overflow

Outro tipo de ataque à memória é o `Buffer Overflow`. Um `Buffer` consiste numa região de memória utilizada para armazenar temporariamente os dados enquanto eles são transferidos de um lugar para outro. Por norma, esta transição é realizada de um input para o `Buffer` ou de um `Buffer` para um output, mas também pode haver o caso de transição entre o `Buffer` para outra variável e vice-versa. Segue um pequeno exemplo da transição realizada com `buffers`.

```
int main(int argc, char const *argv[])
{
    char buffer2[14] = "Over Write Me";
    char buffer1[14];
    char var[5] = "Ola!";

    strcpy(buffer1, var);

    printf("buffer1: %s\n", buffer1);
    printf("buffer2: %s\n", buffer2);

    return 0;
}
```

Por outro lado, o `Buffer Overflow` acontece quando o programa escreve além do espaço alocado em memória para o `buffer`. A principal característica deste `exploit` é conseguir alterar variáveis para que seja possível aceder a zonas de memória ao qual o programa, inicialmente, não estaria programado para aceder.

3.3 Stack Overflow

À semelhança dos sub-capítulos anteriores, é importante diferenciar dois conceitos, `stack` de `Stack Overflow`. A `stack` é um mecanismo de uso de memória que permite ao sistema de memória guardar dados temporariamente do tipo LIFO.

`Stack Overflow` refere-se quando "rompemos" com a `stack`, isto significa, a informação que guardamos é maior que a capacidade de armazenamento da stack, tentando alterar uma zona de memória que não lhe é permitida, originando assim *"segmentation fault"*.

3.3.1 Denial of Service (DoS)

O `exploit` do `Stack Overflow` pode ter diversas finalidades maliciosas, das quais se salienta o ataque DoS. Este é um ataque malicioso que tem como objetivo enviar uma grande quantidade de dados para interromper um serviço de um dispositivo, por exemplo desligar a conexão à Internet.

Este ataque implica que exista um `Stack Overflow`. Durante o mesmo, ao tentar processar uma grande quantidade de informação, o sistema pode, ficar mais lento, deixar de funcionar propriamente e abortar ou ter outros comportamentos maliciosos.

4 Heap Overflow vs Stack Overflow

Como dito anteriormente, a `heap` é uma região de memória onde se guardam, dinamicamente, variáveis. Existem dois grandes tipos de `heap exploits`. No primeiro tipo, aloca-se espaço na memória continuamente, até que não exista mais espaço para alocar e, assim, tenta aceder a uma zona de memória que não tem permissão (neste caso o `Kernel`). O segundo tipo consiste em alocar uma variável com uma dimensão maior que o seu espaço de armazenamento, o que permite ter o mesmo resultado que o primeiro `exploit`, levando ambos a um `Heap Overflow`.

A `stack` é uma região de memória que guarda variáveis locais, parâmetros de uma função e o endereço para retornar. O `exploit`, `Stack Overflow`, tem duas grandes formas de ser realizado, semelhantes aos `exploits` no `head`. Na primeira, declara-se uma variável com um tamanho tão grande que excede o espaço de armazenamento da `stack`. A segunda forma, e usando o exemplo de uma função recursiva, a `stack` guarda a informação previamente dita, sobre cada função, e, ao chamar um elevado número de funções, acaba por tentar escrever valores fora da memória (neste caso a zona de Data), levando ambas a um `Stack Overflow`.

5 Buffer Overflow vs Stack Overflow

Numa forma popular, o significado de `Buffer Overflow` sombreia o `Stack Overflow`, de tal forma que a sua menção é muitas vezes confundida com o `Buffer Overflow`.

O `Buffer Overflow` ocorre, normalmente, quando se tenta aceder a zonas de memória além dos limites. A `Stack Overflow` é quando se excede o espaço de armazenamento.

6 Como evitar

Não existe, por assim dizer, uma técnica universal para resolver ou evitar um `Stack Overflow`, mas sim práticas que podemos aplicar, ou utilizar uma linguagem de programação diferente de modo a diminuir a ocorrência.

É importante diminuir o número de variáveis locais, visto estas serem guardadas na `stack`, ter o cuidado de optimizar as funções recursivas usando *tail-recursion* ou evitar de todo se existem suspeitas de *overflow*. Apesar de ser uma boa prática dividir o programa em várias pequenas funções, não é boa prática dividir demasiado, é necessário existir um equilíbrio.

A linguagem de programação usada também influência a ocorrência de `Stack Overflows`, numa linguagem de baixo nível como o C, requer mais atenção para que este problema, ainda que, versões mais recentes de compiladores, como por exemplo o gcc, já estejam equipados com medidas de prevenção para acessos indevidos a posições e memória. Uma nova linguagem de programação `RUST` que é basicamente a mistura da velocidade de uma linguagem de baixo nível, com a segurança de uma linguagem de alto nível. Apresenta várias vantagens, como a garantia de que não há leitura de espaços de memória não inicializados e que os limites da memória não são ultrapassados.

7 Dificuldades

Houve uma certa dificuldade na pesquisa do tema proposto, pois a presença do site Stackoverflow, direcionava muito a pesquisa, apesar de ser um site de perguntas e respostas sobre programação.

A mistura de informação que existe sobre `Buffer Overflow` e `Stack Overflow`, tornou difícil a compreensão entre os diferentes tipos de `exploits`, apesar de serem muito semelhantes não são iguais, como explicado nos capítulos anteriores.

8 Conclusão

Em suma, apesar de o `Stack Overflow` originar um DoS que parece inicialmente inofensivos, isto é, não executa código malicioso nem tenta aceder ao `Kernel`, se for direcionado a grandes serviços, mesmo que o *downtime* seja de apenas alguns minutos, pode proporcionar grandes prejuízos, dependendo da implementação do serviço.

Por fim, consideramos que este trabalho se encontra bem sintetizado, organizado e simples, demonstrando objetivamente os conceitos e conteúdos interiorizados no decorrer das aulas.