Projeto de programação 2

Cibersegurança - 2023/1 - Prof. Michele Nogueira

Universidade Federal de Minas Ferais

Luís Felipe Ramos Ferreira - 2019022553

Link para o repositório

- Parte 1 - *Exploits* dos alvos 1 e 2

  A parte 1 do projeto de programação 2 consistiu em escrever *exploits* que tirassem proveito das vulnerabilidades presentes nos alvos 1 e 2. Em particular, os alvos em questão são códigos escritos em C que contêm vulnerabilidades devido à possibilidade de causar um *buffer overflow*.

  No alvo 1, a função *strcpy* é utilizada para armazenar o conteúdo da variável *arg* em *out*. Como *strcpy* não faz nenhuma checagem de limite de tamanho de *string* de cópia, o que permite um *overflow* no momento da cópia. Essa vulnerabilidade permite que um *shellcode* seja introduzido como entrada para o *script* e, assim, seja possível ter acesso à um *shell root*.

  No alvo 2, o problema está na implementação da função auxiliar *nmemcpy*. Mais especificamente, na função que exerce a cópia do conteúdo de uma *string* em outra, apesar de existir uma checagem de tamanho das *strings* de cópia, o laço *for* possui um *typo*. A condição de limite imposta no laço é **i <= len** e não **i < len**. Desse modo, um *byte* a mais é sempre lido na cópia da de uma *string* em outra, o que, mais uma vez, permite que o *buffer overflow* seja explorado e um *shellcode* que concede acessos privilegiados a um *shell root* seja possível.

- Parte 2 - Alvos 3, 4, e 5

  O alvo 3 parece conter mais um caso de vulnerabilidade devido a *overflow*, mas dessa vez relacionado a números inteiros, que pode ser exlorado na função *foo*. Em particular, na função, há uma checagem, antes da cópia, se a variável *count* é menor do que a constante definida *MAX_WIDGETS*. No entanto, o valor de *count* pode ser manipulado para que essa condição seja aceita mesmo quando não deveria. Uma estratégia seria utilizar dos conceitos de tipos de inteiros com e sem sinais em C para alterar o tamanho do *buffer* e assim inserir o *shellcode* no programa.

  O alvo 4 sofre da mesma vulnerabilidade do alvo 2. O *typo* presente no laço *for* feito na função auxiliar de cópia de *strings* permite que um *byte* extra seja adicionado na cópia. Ademais, a manipulação dos ponteiros da função *foo*, após o *buffer overflow*, pode permitir que o ponteiro de execução seja direcionado para o *shellcode* já citado.

  Para explorar o alvo 5, utilizei o *script find_gadgets.py* para explorar os

gadgets em potencial que permitiriam explorar o *shellcode*. Os resultados foram salvos em um arquivo para análise, mas não obtive sucesso em encontrar a vulnerabilidade para por em prática. No entanto, na teoria, a ideia se resume em encontrar a posição de memória vulnerável que permite manipular o código e executar o *shellcode*.

- Parte 3 - Vulnerabilidades em um programa do mundo real (bdstar)

  Após utilizar o *GDB*, o resultado da análise dos dois *crashes* encontrados foram os seguintes:

  Podemos ver que ambas aconteceram no arquivo *libarchive/archive_read_support_filter_compress.c* a falha ocorrida foi de *Segmentation fault*, que ocorreu provavelmente ao tentar dereferenciar o ponteiro *state*, que provavelmente não é acessível, e por isso o erro e a consequente vulnerabilidade.

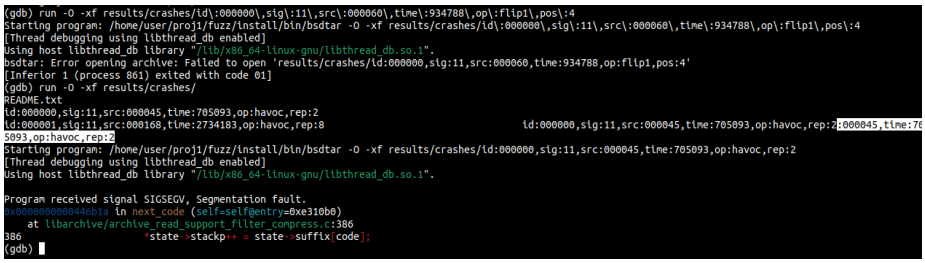

Figure 1: *Crash* 1

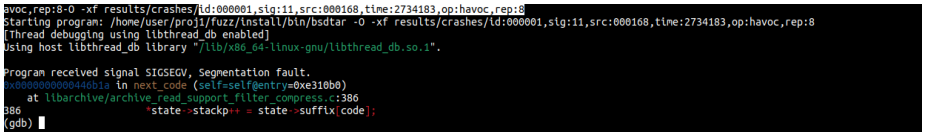

Figure 2: *Crash* 2

O *backtrace* do *GDB* pode ser visto abaixo:

O *link* com redirecionamento para o repositório público no *GitHub* está disposto nos *headers* deste arquivo.

Figure 3: *Backtrace*