Buffer overflow

Esta atividade consiste em desenvolver o experimento Buffer overflow vulnerability do SEED Project.

Antes de desenvolver esse experimento, recomenda-se estudar a seguintes leituras preparatórias:

- Smashing the Stack for Fun and Profit, Aleph One, 1996 (versão PDF)
- Smashing the Stack in 2010, Graziano & Cugliari, 2010
- Smashing the Stack in 2011, Makowski, 2011

No relatório, descreva as atividades efetuadas e explique como funcionam os seguintes mecanismos de proteção:

- Técnica ASLR (*Address Space Layout Randomization*)
- Bit NX (*No eXecute bit*)
- Proteção de pilha oferecida pelo compilador GCC
- Proteção de execução SUID oferecida pelo shell `bash`
- Proteção de execução SUID oferecida pela montagem de partições (comando `mount`)

Roteiro sugerido

Este roteiro foi baseado no experimento de buffer overflow do Seed Project. Os comandos abaixo foram testados em plataforma Ubuntu 13.04 32 e 64 bits.

Deve ser usado como informação complementar, não como substituto ao roteiro original!

Desligar a proteção ASLR (*Adress Space Layout Randomization*) do *kernel*:

```
sudo sysctl -w kernel.randomize_va_space=0
```

Compilar `stack.c` sem a proteção de pilha do compilador:

```
cc stack.c -o stack -fno-stack-protector -g
```

Desligar o flag NX (No Executable Stack) no cabecalho do executável ELF:

```
sudo apt-get install execstack
execstack stack
execstack -s stack
execstack stack
```

Ajustar usuário e permissões do executável `stack`:

```
sudo chown root.root stack
sudo chmod 4755 stack
```

Gerar um `badfile` qualquer para poder executar `stack` (sem *overflow*):

```
echo "um teste..." > badfile
```

Depurar `stack` para descobrir o endereço do ponteiro `str`:

```
gdb stack
   br 13           // insere breakpoint na linha 13 (strcpy)
   run             // inicia a execução
   print /x str    // obtém o valor do ponteiro str
   quit            // encerra o debugger
```

Usar o valor obtido do ponteiro `str` para ajustar `exploit.c`:

- compreenda a estrutura da pilha do programa `stack.c` dentro da chamada a `bof()`
- descubra que posição da variável `buffer` corresponde ao endereço de retorno da função `bof`
- escreva o código necessário em `exploit.c` para produzir um arquivo `badfile` que carregue o *shellcode* na memória do alvo e sobrescreva o endereço de retorno da função `bof` com o endereço do *shellcode*
- Para ver o conteúdo de `badfile` use o comando hd (*hex dump*)

Compilar e executar `exploit.c` para gerar o arquivo `badfile` malicioso:

```
cc exploit.c -o exploit
./exploit
```

Usar o `badfile` malicioso para explorar a vulnerabilidade de `stack`:

```
./stack
whoami
```

Se tudo estiver certo, o último comando deve retornar um *prompt* de root (#) e o comando `whoami` deve informar `root`. Se não funcionar, verificar:

- se estiver em uma partição de disco montada com o flag `nosuid` (verificar isso com o comando `mount`), mover os arquivos `stack` e `badfile` para `/tmp`
- se a arquitetura do *shellcode* está correta (32 ou 64 bits).

No caso de arquiteturas de 64 bits, este site explica com detalhes a construção de um *shellcode* para processadores de 64 bits no padrão x86_64 (Intel/AMD). Outros *shellcodes* podem ser encontrados aqui.

Contornar a proteção ASLR

Religar a proteção ASLR do kernel:

```
sudo sysctl -w kernel.randomize_va_space=2
```

Executar o ataque continuamente, para tentar contornar a proteção ASLR (pode demorar muito!):

```
while true ; do ./stack ; done
```

From:
https://wiki.inf.ufpr.br/maziero/ - **Prof. Carlos Maziero**

Permanent link:
https://wiki.inf.ufpr.br/maziero/doku.php?id=sc:buffer_overflow

Last update: **2017/04/13 15:34**