

MC504/MC514 - Sistemas Operacionais

Buffer Overflow

Islene Calciolari Garcia

Instituto de Computação - Unicamp

Primeiro Semestre de 2014

Sumário

- 1 Introdução
- 2 Alterar o endereço de retorno
- 3 Produzir o shell code
- 4 Analisar o execve
- 5 Arrumar a chamada para o execve
- 6 Explorando a vítima
- 7 Proteção

Buffer Overflow

- Ataque que insere mais dados do que o espaço previamente alocado. Estes dados extras na verdade são trechos de códigos que serão executados pelo programa-vítima.
- Esta aula foi baseada na série Buffer Overflow Primer, de Vivek Ramachandran

Registradores Intel 32-bits

- EBP: *base pointer*
- ESP: *stack pointer*

- EAX: acumulador
- ECX: contador
- EDX: dados
- ESI: *source index*
- EDI: *destination index*

Vamos examinar a pilha de execução

- Componentes do frame (não necessariamente nesta ordem):
 - valor de retorno
 - endereço de retorno
 - registradores
 - argumentos para a função
 - variáveis locais da função
- Veja os códigos `pilha.c` e `pilha2.c`

Funções vulneráveis

- Veja `man gets()`
- Veja o código `buffer.c`
- Insira um buffer maior do que o esperado e veja o resultado
- Veja o código `buffer-strncpy.c`

Como colocar o novo ponto de execução via entrada padrão?

- Como colocar encontrar e inserir o endereço na string?
- Veja man printf()

```
$ printf "abcdef\x01\x02\x03\x04" | ./buffer
```

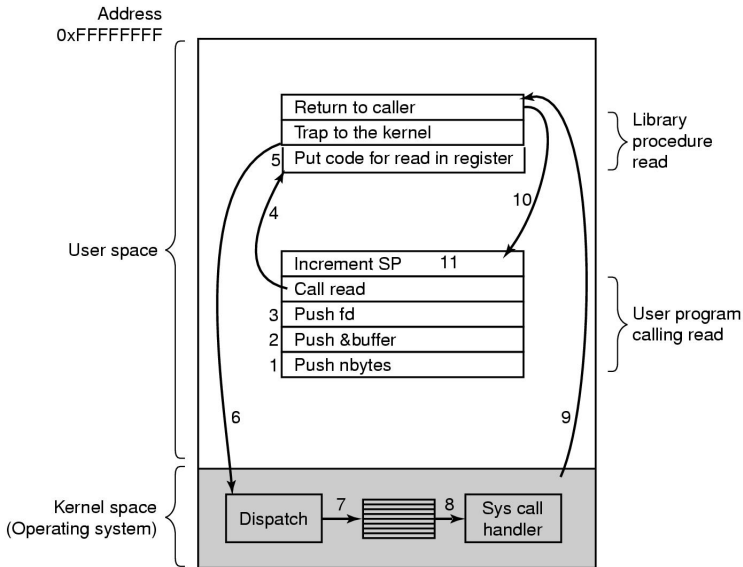
Produzir o shell code

- Código malicioso a ser inserido tem este nome porque, na maioria das vezes, o código adicionado iria abrir uma nova shell.
- Vamos escrever armadilhas. A vítima vai colaborar nos testes executando código via mmap.
- Veja os códigos map-armadilha e map-vitima
- Tente modificar map-armadilha para chamar uma função tipo printf e uma função qualquer.

Chamadas de sistema

- Fazem a fronteira entre o modo usuário e o modo kernel (protegido)
- Exemplos: fork, waitpid, execve, open, close, read, write, mkdir, link, unlink, ...
- O kernel deve ter uma tabela com as várias chamadas.

Chamada de sistema



execve

- Veja o exemplo `execve.c`

```
int main() {  
    char *args[2] = {"/bin/bash", NULL};  
  
    execve(args[0], args, NULL);  
    return 0;  
}
```

Veja `map-armadilha-execve.c`

Todos os argumentos devem estar na pilha

- Argumentos
 - `filename`
 - `argv`
 - `envp`
- E a string de entrada não pode ter NULLs.
- Veja o código `map-armadilha-execve2.c`

Explorando a vítima

- Shell code está preparado.
- Como saber em qual endereço o shell code está posicionado?
- NOP Sled antes do shell code permite a recuperação de pequenas falhas no cálculo do endereço.
- Ataque força-bruta?

Proteção das páginas

- Controlar quais páginas são executáveis poderia resolver o problema?
- Return to libc

Como funciona o Stack Canary

- Termo canário inspirado nos pássaros utilizados nas minas de carvão.
- Valores chave são inseridos na pilha
- Caso esses valores sejam alterados, o programa deve interromper sua execução.
- Recompile os exemplos sem a flag `-fno-stack-protector`