Sistemas Operacionais

Rodrigo Rubira Branco
rodrigo@kernelhacking.com
rodrigo@fgp.com.br

Basico de Shell Script

- Declaracao de Variaveis
- If
- while
- for
- Execucao de comandos
- cut, grep

- TRABALHO em Sala

Desenvolvendo modulos de Kernel Linux v2.4

```
/* Prototipo de um modulo */
#define __KERNEL__
#define MODULE
#define LINUX
#include <linux/module.h>
#include <linux/kernel.h>
MODULE_LICENSE("GPL");

int init_module(void) {
        printk("Modulo carregado...");
        return 0;
}

void cleanup_module(void)
{
        printk("Modulo descarregado...");
}
```

- init_module()

Chamada ao carregar o modulo, equivalente a funcao main() de um programa em C comum.

- cleanup_module()

Chamada ao se descarregar o modulo, deve realizar todas as funcoes de “limpeza”, ou seja, voltar os parametros modificados pelo Kernel do Linux.

- MODULE_LICENSE(“GPL”)

Responsavel por estabelecer o licenciamento definido para este modulo. Projetos existem que envolvam permitir maior acesso a interfaces do kernel por modulos que declarem ser GPL.

- printk()

Substituto ao printf() quando se programa para kernel.

- Compilacao

gcc -I/lib/modules/`uname -r`/build/include -c modulo.c

- Inserindo o modulo no Kernel

insmod modulo.o

- Listando modulos do Kernel

lsmod

- Removendo modulos do Kernel

rmmod modulo -> Sem o .o no final

- Diferencas interessantes (Recursos do Kernel)

- Otimizacoes de testes/condicoes

```
if ( unlikely(condicao_que_nao_ocorrera) )
{

}
else
{

}
```

- Passando parametros para um modulo

```
unsigned long endereco; char *arquivo; int teste;
MODULE_PARM(endereco,”l”);
MODULE_PARM(arquivo,”s”);
MODULE_PARM(teste,”i”);
```

- Revisitando as chamadas ao sistema

```
#include <sys/syscall.h>

extern void * sys_call_table[];

/* Iremos armazenar o endereço da system call original */
int (*original_sys_exit)(int);

/* Criamos aqui uma funçao a ser chamada quando alguém fizer uma chamada para exit */
int our_fake_exit_function(int error_code)
{
/* Imprimir mensagem e o codigo passado para exit */
printk("codigo de erro=%d\n",error_code);

/* Executamos aqui a system call original */
return original_sys_exit(error_code);

}
```

```
/* Iremos armazenar o endereço da system call original */
int (*original_sys_exit)(int);

/* Criamos aqui uma funçao a ser chamada quando alguém fizer uma chamada
        para exit
*/
int our_fake_exit_function(int error_code)
{
/* Simplesmente imprimimos uma mensagem e o código recebido */
printk("hey, a system call sys_exit foi chamada com o codigo de erro=%d\n",error_code);

/* Executamos aqui a system call original */
return original_sys_exit(error_code);

}
```

```
/* Nossa nova init_module */
int init_module(void)
{
        /* Armazenar o endereco original  */
        original_sys_exit=sys_call_table[SYS_exit];

        /* Substituir o endereco pela funcao hackeada  */
        sys_call_table[SYS_exit]=our_fake_exit_function;

        /* Alertamos que o módulo foi adequadamente carregado */
        printk("\nMódulo carregado, SYS_exit hackeado\n");
        return 0;
}
/* Nosso novo cleanup_module */
void cleanup_module(void)
{
        /* Retornamos o valor original  */
        sys_call_table[SYS_exit]=original_sys_exit;

        printk("\n SYS_exit hackeada removida\n");
}
```

FIM! Será mesmo?

DÚVIDAS?!?

Rodrigo Rubira Branco
rodrigo@kernelhacking.com