Sistemas Operacionais B - Engenharia de Computação

Projeto 2

Participantes	RA
Bruno Vicente Donaio Kitaka	16156341
Paulo Jansen de Oliveira Figueiredo	16043028
Rafael Fioramonte	16032708
Raissa Furlan Davinha	15032006
Vinicius Trevisan	16011231

Sumário

1. Introdução:

Esse projeto teve como intuito explorar em detalhes o desenvolvimento e aplicação de um módulo de kernel como sistema de arquivo que utiliza a API criptográfica do kernel Linux. O estudo visa o desenvolvimento do código do módulo, compilação do mesmo, instalação como novo módulo do kernel da máquina e utilização dele em um programa em espaço de usuário.

O módulo deve receber uma chave de criptografia ao ser inicializado no kernel e, ao ser utilizado em um programa em espaço de usuário, deve passar a operação a ser realizada: "1" para criar um novo arquivo e "2" para atualizar um arquivo existente. Ao escolher a opção, passa-se o caminho onde o arquivo será criado ou editado e uma string que, após recebida pelo programa em espaço de usuário, será gravada no arquivo pelo módulo criado.

O módulo de arquivos desenvolvido recebe os dados ao realizar funções de escrita ou leitura, o módulo então deverá criptografar os dados e gravar no arquivo em caso de uma operação de escrita ou descriptografar e ler do arquivo no caso de uma operação de leitura.

A criptografia e descriptografia devem utilizar o algoritmo AES (*Advanced Encryption Standard*, uma especificação para criptografia de dados estabelecida pelo Instituto Nacional de Padrões e Tecnologia) em modo ECB (*Electronic CodeBook*, um modo de operação de criptografia simples não encadeada). Para a utilização desse módulo, deve ser executado um programa em espaço de usuário que realize funções de escrita com a string desejada, para leitura se utiliza a função "*hd* nome-do-arquivo" para mostrar o conteúdo do arquivo em hexadecimal.

2. Fundamentação Teórica:

- **Criptografia e Descriptografia:** A criptografia consiste em converter informações comuns em texto ininteligível. A decriptação, por outro lado, é o inverso, em outras palavras, passar o texto cifrado ininteligível de volta para texto puro. Na maioria dos casos, uma chave conhecida é utilizada para realizar ambas as tarefas de modo a garantir a segurança das informações, essa, tendo que ser devidamente armazenada, para impossibilitar o acesso aos dados por terceiros.
- **Algoritmo AES:** AES entende-se como uma especificação para criptografia simétrica ou assimétrica podendo ser de 128, 192 ou 256 bits, esse sendo o tamanho máximo para a chave de entrada. É uma cifra por blocos, ou seja, a entrada e saída são limitadas conforme o tipo de especificação utilizada. Dessa forma, para realizar a criptografia ou descriptografia de uma grande quantidade de dados, este algoritmo deve ser executado inúmeras vezes.
- **Sistema de arquivo:** controla como os dados são armazenados e recuperados, possibilitando ao sistema operacional decodificar os dados armazenados e lê-los ou gravá-los.
- **Minix:** sistema operacional *UNIX-like* desenvolvido por *Andrew S. Tanenbaum* com um objetivo educativo, o seu sistema de arquivos foi utilizado no projeto.

3. Desenvolvimento:

Os arquivos do sistema Minix foram baixados junto dos arquivos do kernel, igualmente feito no projeto passado. A fase inicial do projeto teve o propósito de entendimento do sistema de arquivo utilizado para encontrar em qual ponto se faz a escrita e leitura em arquivo, para isso foi necessário inicialmente modificar os códigos de arquivo Minix encontrados, colocando funções de *print* para se entender o fluxo de funcionamento do sistema de arquivos. Após o entendimento e identificado as funções de escrita e leitura iniciamos a modificar as mesmas com o objetivo de incluir a função de criptografia, antes de gravar para a função write e a de descriptografia após a leitura para a função read. O alvo encontrado foi dentro do arquivo file.c, a estrutura *minix_file_operations*. Dentro da estrutura, substituímos as funções genéricas por funções criadas, interferindo no ciclo do arquivo, para adicionar a criptografia.

Na nova função relacionada à escrita, a mesma cifra o conteúdo enviado, conforme foi feito no projeto anterior, e depois chama a função genérica que foi interceptada.

Similar foi feito com a função de leitura, também interceptada, mas esta recebe a função de descriptografar os dados, e depois devolve à função genérica. Por fim foi criado um programa em espaço de usuário que faz a escrita de um conteúdo especificado no arquivo utilizando *fwrite*.

4. Resultados:

A compilação do módulo minix se deu por um makefile chamado por um script, o script também monta os arquivos imagem para ser utilizados em testes. Nas imagens abaixo os passos foram feitos manualmente para demonstrar as saídas de cada operação no terminal.

```
root@vm:/home/paulo/Desktop/minix# make -C /lib/modules/$(uname -r)/build M=$(pwd) modules
make: Entering directory '/usr/src/linux-headers-4.15.0-39-generic'
  CC [M]  /home/paulo/Desktop/minix/bitmap.o
  CC [M]  /home/paulo/Desktop/minix/itree_v1.o
  CC [M]  /home/paulo/Desktop/minix/itree_v2.o
  CC [M]  /home/paulo/Desktop/minix/namei.o
  CC [M]  /home/paulo/Desktop/minix/inode.o
/home/paulo/Desktop/minix/inode.c: In function ‘init_minix_fs’:
/home/paulo/Desktop/minix/inode.c:697:2: warning: ISO C90 forbids mixed declarations and code [
-Wdeclaration-after-statement]
  int err = init_inodecache();
  ^
  CC [M]  /home/paulo/Desktop/minix/file.o
/home/paulo/Desktop/minix/file.c: In function ‘mitm_write_iter’:
/home/paulo/Desktop/minix/file.c:80:30: warning: initialization discards ‘const’ qualifier from
 pointer target type [-Wdiscarded-qualifiers]
   struct iovec *modificado = iov_it->iov;
                              ^
  CC [M]  /home/paulo/Desktop/minix/dir.o
  LD [M]  /home/paulo/Desktop/minix/minix.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/paulo/Desktop/minix/minix.mod.o
  LD [M]  /home/paulo/Desktop/minix/minix.ko
make: Leaving directory '/usr/src/linux-headers-4.15.0-39-generic'
root@vm:/home/paulo/Desktop/minix#
```

Figura 1: Compilando o dispositivo Minix.

```
root@vm:/home/paulo/Desktop/Projeto-2-SO-B-dev# dd if=/dev/zero of=teste.img bs=1k count=10000
10000+0 records in
10000+0 records out
10240000 bytes (10 MB, 9,8 MiB) copied, 0,0257797 s, 397 MB/s
root@vm:/home/paulo/Desktop/Projeto-2-SO-B-dev# losetup /dev/loop0 teste.img
root@vm:/home/paulo/Desktop/Projeto-2-SO-B-dev# mkfs -t minix /dev/loop0
3360 inodes
10000 blocks
Firstdatazone=110 (110)
Zonesize=1024
Maxsize=268966912

root@vm:/home/paulo/Desktop/Projeto-2-SO-B-dev# mkdir /mnt/point1
root@vm:/home/paulo/Desktop/Projeto-2-SO-B-dev# mount -t minix /dev/loop0 /mnt/point1
root@vm:/home/paulo/Desktop/Projeto-2-SO-B-dev#
```

Figura 2: Criando e montando o dispositivo Minix.

Com a imagem montada e o módulo compilado o mesmo é carregado e em seguida utilizamos o comando dmesg para observar o retorno ao ser inserido.

```
root@vm:/home/paulo/Desktop/minix# insmod minix.ko key="abcd"
root@vm:/home/paulo/Desktop/minix# lsmod | grep minix
minix                  40960  0
root@vm:/home/paulo/Desktop/minix#
```

Figura 3: Carregando o módulo Minix.

```
[  238.662360] minix: loading out-of-tree module taints kernel.
[  238.663032] Minixmodule: Modificacao file.c (criptografia) e inode.c (chave)
[  238.663033] Minixmodule: Chave (Key) BRUTA recebida: abcd
[  238.663035] Minixmodule: Chave (Key) CONSIDERADA em hexadecimal: abcd
[  238.663035] Minixmodule: inode.c init_minix_fs
```

Figura 4: dmesg, carregando módulo.

O programa em espaço usuário foi utilizado para os testes de escrita possuindo as funcionalidades de criar e atualizar.

```
Teste Minixmodule Projeto 2
.::MENU::.
1 - Criar
2 - Atualizar
0 - Sair
Selecione sua opcao: 1
```

Figura 5: Interface do programa de teste em espaço de usuário.

```
Digite o caminho do arquivo a ser aberto: /mnt/point1/teste
Opcao 1: Criar Arquivo
Digite o que deseja escrever no arquivo /mnt/point1/teste: vinicius
.::MENU::.
1 - Criar
2 - Atualizar
0 - Sair
Selecione sua opcao:
```

Figura 6: Programa de teste em espaço de usuário salvando a string “vinicius” no arquivo.

Após criado o arquivo com a string fornecida pela opção 1 testamos a leitura do arquivo com o código minix modificado para testar a função de leitura com descriptografia e também testamos a leitura utilizando o código minix vanilla para testar a criação e criptografia.

```
root@vm:/mnt/point1# hd teste
00000000  76 69 6e 69 63 69 75 73  00 00 00 00 00 00 00 00  |vinicius........|
00000010
```

Figura 7: Opção 1, conteúdo do arquivo utilizando minix modificado.

```
00000000  ef dc d0 d4 29 6c 2e 4d  f9 d6 47 95 cc fc 96 7a  |....)l.M..G....z|
00000010
```

Figura 8: Opção 1, conteúdo do arquivo utilizando minix vanilla.

A função de “atualizar” do código em espaço de usuário lê o conteúdo do arquivo e mostra ao usuário pedindo que ele faça as alterações conforme desejado.

Feito ambos os testes (com minix modificado e com vanilla) os resultados demonstram que a criptografia e decriptografia são realizadas com sucesso

```
Digite o caminho do arquivo a ser aberto: /mnt/point1/teste
Opcao 1: Criar Arquivo
Digite o que deseja escrever no arquivo /mnt/point1/teste: vinicius
.::MENU::.
1 - Criar
2 - Atualizar
0 - Sair
Selecione sua opcao: 2
```

Figura 9: Opção

```
Digite o caminho do arquivo a ser aberto: /mnt/point1/teste
Opcao 2: Atualizar Arquivo
***ATENÇÃO: SEU ARQUIVO SERÁ REESCRITO***
Dado lido do arquivo /mnt/point1/teste: vinicius
O que deseja escrever no arquivo: Vinicius
```

Figura 10: Opção 2 com minix modificado substituindo a string “vinicius” por “Vinicius”.

```
root@vm:/mnt/point1# hd teste
00000000  56 69 6e 69 63 69 75 73  00 00 00 00 00 00 00 00  |Vinicius........|
00000010
```

Figura 11: Opção 2, conteúdo do arquivo utilizando minix modificado.

```
Digite o caminho do arquivo a ser aberto: /mnt/point1/teste
Opcao 2: Atualizar Arquivo
***ATENÇÃO: SEU ARQUIVO SERÁ REESCRITO***
Dado lido do arquivo /mnt/point1/teste: p"◆D#o3MINh\Q*
O que deseja escrever no arquivo:
```

Figura 12: Opção 2 com minix vanilla substituindo a string “vinicius” por “Vinicius”.

```
root@vm:/mnt/point1# hd teste
00000000  0e 70 22 bd 44 23 6f 33  4d 49 4e 68 5c 51 2a 16  |.p".D#o3MINh\Q*.|
00000010
```

Figura 13: Opção 2, conteúdo do arquivo utilizando minix vanilla.

5. **Conclusão:** Após testes alternando entre o código minix modificado e o minix sem alterações pudemos concluir que o nosso módulo estava funcionando corretamente como desejado. Nesse projeto aprendemos a utilizar técnicas de engenharia reversa e funções de interceptação para criar nosso próprio módulo de criptografia de disco.