



Pontifícia Universidade Católica de Campinas
Faculdade de Engenharia de Computação -
FECOMP

Sistemas Operacionais B – Relatório Experimento 2
Chamada de sistema criptográfica no kernel Linux

Beatriz Morelatto Lorente RA: 18071597

Cesar Marrote Manzano RA: 18051755

Fabício Silva Cardoso RA: 18023481

Pedro Ignácio Trevisan RA: 18016568

Sumário

1. Introdução.....	3
2. Algoritmo de criptografia utilizado.....	4
3. Fluxo das informações.....	5
4. Passos para implementação de uma syscall.....	6
5. Resultados.....	8
6. Conclusão.....	11

Introdução

O experimento desenvolvido pretende demonstrar os passos feitos para o desenvolvimento de chamadas de sistema (system calls) que podem armazenar e ler arquivos usando a API criptográfica do kernel. Foram feitas duas chamadas de sistema:

- Na primeira system call, o usuário envia uma mensagem que é criptografada e depois gravada no arquivo
- Na segunda system call, a mensagem armazenada no arquivo é lida, decifrada e enviada para o usuário.

Para a cifragem e decifragem da mensagem foi usado o algoritmo AES em modo ECB.

Algoritmos de criptografia utilizados

Como citado anteriormente, para cifrar e decifrar a string fornecida pelo usuário foi utilizado o algoritmo AES em modo ECB.

Primeiramente é necessário entender como o algoritmo AES (Advanced Encryption Standard) funciona. O algoritmo consiste em uma criptografia simétrica de blocos de tamanho fixo (utilizamos blocos de 16 bytes, 128 bits). O algoritmo usa uma chave para cifrar e decifrar os blocos e esta também tem 16 bytes.

O modo de criptografia ECB (Eletronic CodeBook), é um dos mais simples que temos, é a primeira geração dos modos AES. A mensagem é dividida em blocos que são criptografados separadamente. Como o modo é bem simples, ele não oferece muita segurança, uma vez que não oculta padrões de dados, não sendo recomendado caso o sistema necessite de uma confiabilidade alta. O esquema abaixo mostra como o algoritmo funciona.

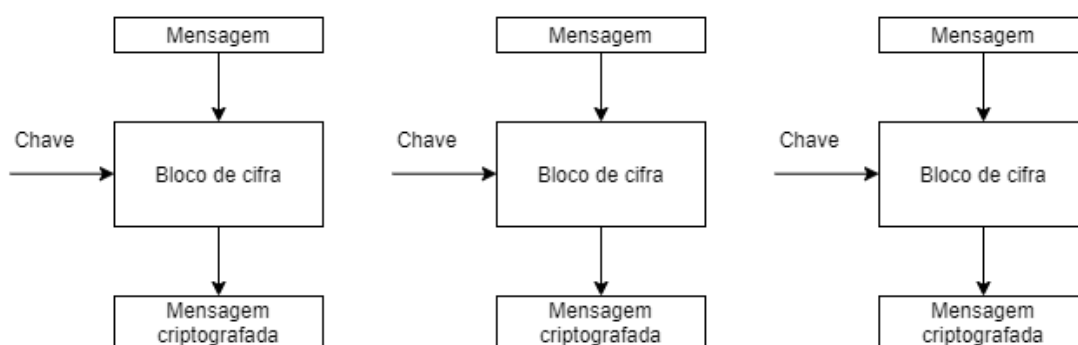


Figura 1 - Funcionamento do algoritmo AES em modo ECB

Fluxo das informações

Para entender como o programa funciona, será mostrado um esquema, em alto nível, mostrando o fluxo das informações. Esse esquema engloba tanto o programa de teste como as chamadas de sistema criadas.

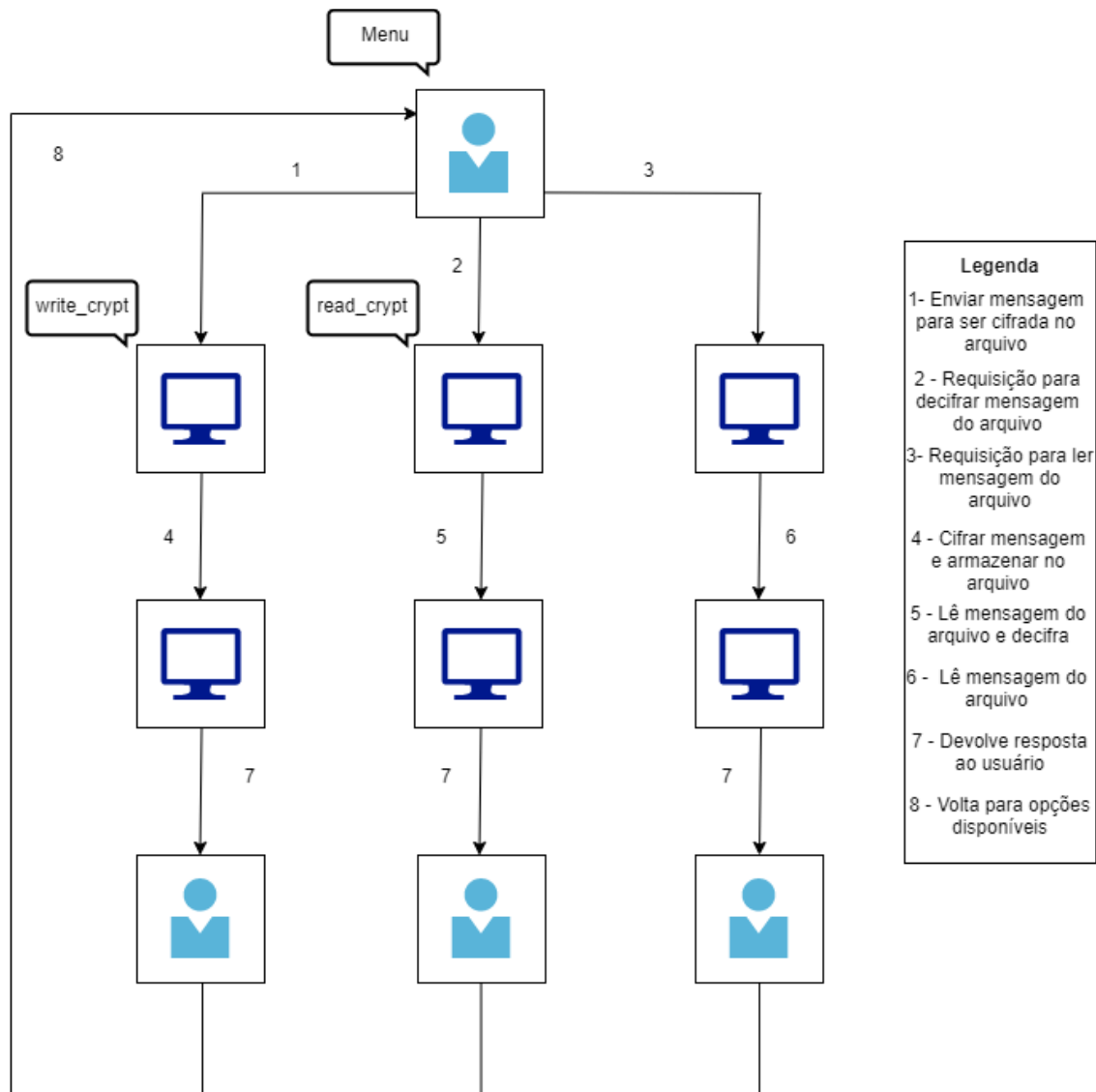
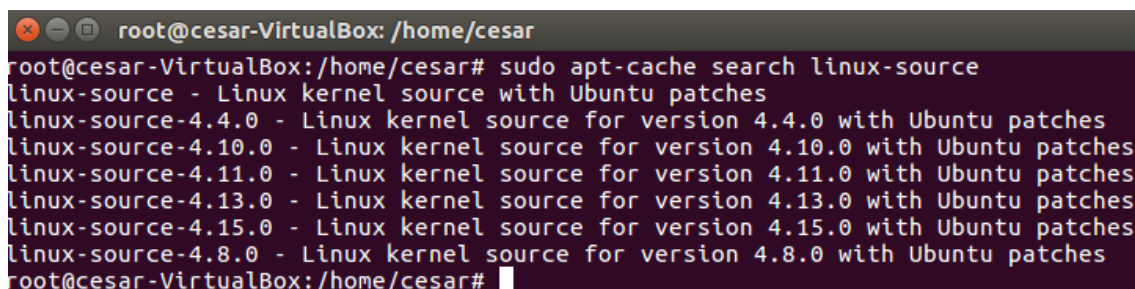


Figura 2 - Fluxo das informações no programa

Passos para implementação de uma syscall

Para implementar uma syscall há alguns passos principais que serão mostrados abaixo.

Para a nossa implementação foi usada a versão do linux 4.15.0. Para instalar essa versão ou outras disponíveis, basta digitar 'sudo apt-cache search linux-source', no terminal e depois 'sudo apt-get install linux-source-[versão desejada]'.



```
root@cesar-VirtualBox: /home/cesar
root@cesar-VirtualBox:/home/cesar# sudo apt-cache search linux-source
linux-source - Linux kernel source with Ubuntu patches
linux-source-4.4.0 - Linux kernel source for version 4.4.0 with Ubuntu patches
linux-source-4.10.0 - Linux kernel source for version 4.10.0 with Ubuntu patches
linux-source-4.11.0 - Linux kernel source for version 4.11.0 with Ubuntu patches
linux-source-4.13.0 - Linux kernel source for version 4.13.0 with Ubuntu patches
linux-source-4.15.0 - Linux kernel source for version 4.15.0 with Ubuntu patches
linux-source-4.8.0 - Linux kernel source for version 4.8.0 with Ubuntu patches
root@cesar-VirtualBox:/home/cesar#
```

Figura 3 - Versões de kernel disponíveis após rodar o comando 'sudo apt-cache search linux-source'

Depois de instalar a versão desejada do kernel do linux, ela estará disponível na pasta '/usr/src'.

Acesse a pasta da sua versão do kernel e crie um novo diretório, que conterá todos os arquivos da sua syscall. Para criar o diretório, digite 'mkdir [nome do diretório]' e depois entre com o comando 'cd [nome do diretório]'.

Após criar um ou mais arquivos que conterá sua(s) syscall(s), é necessário criar um Makefile dentro do diretório, contendo o conteúdo 'obj-y:=[nome do arquivo].o'. Esse arquivo será responsável por compilar sua chamada de sistema.

Com o Makefile criado é necessário modificar outro Makefile, mas desta vez do kernel. Volte para a pasta principal do kernel e edite-o. Procure pelo nome 'core-y' e adicione o nome do diretório criado no final da linha, como mostrado abaixo.

```
PHONY += prepare0

ifeq ($(KBUILD_EXTMOD),)
core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ wrCrypt/

vmlinux-dirs := $(patsubst %/,%, $(filter %/, $(init-y) $(init-m) \
$(core-y) $(core-m) $(drivers-y) $(drivers-m) \
$(net-y) $(net-m) $(libs-y) $(libs-m) $(virt-y)))

vmlinux-alldirs := $(sort $(vmlinux-dirs) $(patsubst %/,%, $(filter %/, \
$(init-) $(core-) $(drivers-) $(net-) $(libs-) $(virt-))))
```

Figura 4 - Modificação do Makefile do kernel

Depois é necessário modificar o arquivo 'syscall_64.tbl', encontrado no caminho '/arch/x86/entry/syscalls'. Depois adicione sua(s) syscall(s), como mostrado abaixo.

```

328      64      pwritev2      sys_pwritev2
329      common pkey_mprotect sys_pkey_mprotect
330      common pkey_alloc   sys_pkey_alloc
331      common pkey_free    sys_pkey_free
332      common statx        sys_statx
333      common write_crypt   sys_write_crypt
334      common read_crypt    sys_read_crypt

#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation.
#
512      x32      rt_sigaction      compat_sys_rt_sigaction
513      x32      rt_sigreturn      sys32_x32_rt_sigreturn

```

Figura 5 - Modificação syscall_64.tbl

Por fim é necessário o arquivo 'syscalls.h', disponível no diretório '/include/linux'. Aqui será feita a declaração do corpo, assinatura, da função, no final do arquivo.

```

asmlinkage long sys_mlock2(unsigned long start, size_t len, int flags);

asmlinkage long sys_pkey_mprotect(unsigned long start, size_t len,
                                   unsigned long prot, int pkey);
asmlinkage long sys_pkey_alloc(unsigned long flags, unsigned long init_val);
asmlinkage long sys_pkey_free(int pkey);
asmlinkage long sys_statx(int dfd, const char __user *path, unsigned flags,
                           unsigned mask, struct statx __user *buffer);
asmlinkage long sys_write_crypt(int fd, const void *buf, size_t nbytes);
asmlinkage long sys_read_crypt(int fd, const void *buf, size_t nbytes);

#endif

```

Figura 6 - Modificação do arquivo 'syscalls.h'

Com tudo pronto é necessário compilar o seu kernel, para que suas modificações sejam salvas. Para isso é necessário rodar inicialmente o comando 'make menuconfig' para modificar o que você quiser no kernel. É importante lembrar que é ininteressante ter um kernel que compile rapidamente, uma vez que toda alteração que for feita na syscall, será necessário compilar o kernel. Depois de configurar o kernel, rode o comando 'make -jN && make modules install -jN && make install -jN && reboot', sendo N o número de CPUs disponíveis na sua máquina virtual. Após esse processo, suas modificações poderão ser testadas.

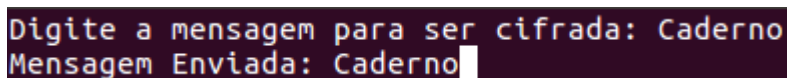
Resultados

Para a apresentação dos resultados obtidos, primeiramente será mostrado o resultado da escrita no arquivo e depois o resultado da leitura do mesmo. Para testar a chamada de sistema, foi utilizado um gerador de palavras aleatórias, que gerando quatro palavras e também escolhemos uma para testar, que são:

- Caderno (1)
- Calcanhar (2)
- Mirante (3)
- Gafanhoto (4)
- Ordem e progresso (5)

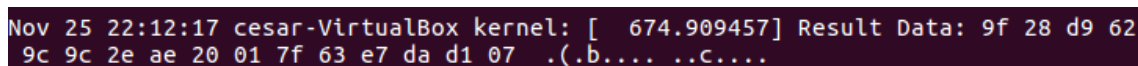
1 - Primeira Palavra

Escrita no arquivo



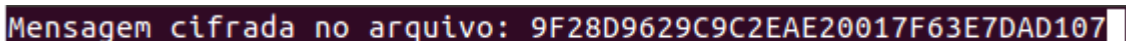
```
Digite a mensagem para ser cifrada: Caderno
Mensagem Enviada: Caderno
```

Figura 7 - Mensagem enviada ao arquivo



```
Nov 25 22:12:17 cesar-VirtualBox kernel: [ 674.909457] Result Data: 9f 28 d9 62
9c 9c 2e ae 20 01 7f 63 e7 da d1 07 .(.b.... ..c....
```

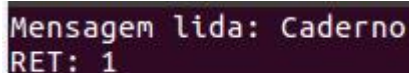
Figura 8 - Visualização da mensagem cifrada utilizando o log do kernel



```
Mensagem cifrada no arquivo: 9F28D9629C9C2EAE20017F63E7DAD107
```

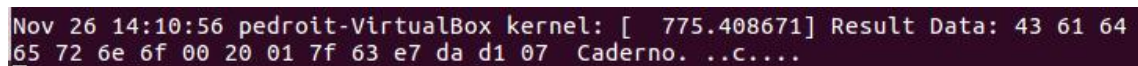
Figura 9 - Mensagem lida do arquivo

Leitura do arquivo



```
Mensagem lida: Caderno
RET: 1
```

Figura 10 - Mensagem lida e decifrada do arquivo



```
Nov 26 14:10:56 pedroit-VirtualBox kernel: [ 775.408671] Result Data: 43 61 64
65 72 6e 6f 00 20 01 7f 63 e7 da d1 07 Caderno. ..c....
```

Figura 11 - Visualização da mensagem decifrada utilizando o log do kernel

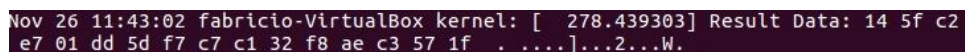
2 - Segunda Palavra

Escrita no arquivo



```
Digite a mensagem para ser cifrada: Calcanhar
Mensagem Enviada: Calcanhar
```

Figura 12 - Mensagem enviada ao arquivo



```
Nov 26 11:43:02 fabricio-VirtualBox kernel: [ 278.439303] Result Data: 14 5f c2
e7 01 dd 5d f7 c7 c1 32 f8 ae c3 57 1f . ....]...2...W.
```

Figura 13 - Visualização da mensagem cifrada utilizando o log do kernel


```
Mensagem cifrada no arquivo: 145FC2E701DD5DF7C7C132F8AEC3571F
```

Figura 14 - Mensagem lida do arquivo

Leitura do arquivo

```
Mensagem lida: Calcanhar  
RET: 1
```

Figura 15 - Mensagem lida e decifrada do arquivo

```
Nov 26 14:13:07 pedroit-VirtualBox kernel: [ 906.376922] Result Data: 43 61 6c  
63 61 6e 68 61 72 00 32 f8 ae c3 57 1f Calcanhar.2...W.
```

Figura 16 - Visualização da mensagem decifrada utilizando o log do kernel

3 - Terceira Palavra

Escrita no arquivo

```
Digite a mensagem para ser cifrada: Mirante  
Mensagem Enviada: Mirante
```

Figura 17 - Mensagem enviada ao arquivo

```
Nov 26 11:47:05 fabricio-VirtualBox kernel: [ 522.176627] Result Data: dd 08 55  
c5 42 c6 c7 69 89 53 4e 9c 42 72 07 6c ..U.B..i.SN.Br.l
```

Figura 18 - Visualização da mensagem cifrada utilizando o log do kernel

```
Mensagem cifrada no arquivo: DD0855C542C6C76989534E9C4272076C
```

Figura 19 - Mensagem lida do arquivo

Leitura do arquivo

```
Mensagem lida: Mirante  
RET: 1
```

Figura 20 - Mensagem lida e decifrada do arquivo

```
Nov 26 15:00:15 pedroit-VirtualBox kernel: [ 318.355868] Result Data: 4d 69 72  
61 6e 74 65 00 89 53 4e 9c 42 72 07 6c 00 00 00 00 00 00 00 00 20 00 00 00 00 00  
00 00 Mirante..SN.Br.l.....
```

Figura 21 - Visualização da mensagem decifrada utilizando o log do kernel

4 - Quarta Palavra

Escrita no arquivo

```
Digite a mensagem para ser cifrada: Gafanhoto  
Mensagem Enviada: Gafanhoto
```

Figura 22 - Mensagem enviada ao arquivo

```
Nov 26 11:50:01 fabricio-VirtualBox kernel: [ 697.740761] Result Data: 88 47 ee  
e0 aa 59 d3 37 4b 46 73 c3 28 9c b8 63 .G...Y.7KFs.(...c
```

Figura 23 - Visualização da mensagem cifrada utilizando o log do kernel

```
Mensagem cifrada no arquivo: 8847EEE0AA59D3374B4673C3289CB863
```

Figura 24 - A string de entrada cifrada

Leitura do arquivo

```
Mensagem lida: Gafanhoto  
RET: 1
```

Figura 25 - Mensagem lida e decifrada do arquivo

```
Nov 26 15:01:16 pedroit-VirtualBox kernel: [ 379.170285] Result Data: 47 61 66  
61 6e 68 6f 74 6f 00 73 c3 28 9c b8 63 00 00 00 00 00 00 20 00 00 00 00 00  
00 00 Gafanhoto.s(...C..... .....
```

Figura 26 - Visualização da mensagem decifrada utilizando o log do kernel

5 - Quinta Palavra

Escrita no arquivo

```
Digite a mensagem para ser cifrada: Ordem e progresso  
Mensagem Enviada: Ordem e progresso
```

Figura 27 - Mensagem enviada ao arquivo

```
Nov 26 11:52:08 fabricio-VirtualBox kernel: [ 825.023636] Result Data: cb 54 b4  
8a 88 69 6d 1d 19 fd fd 0b 1d 1b 7f d1 .T...im.....
```

Figura 28 - Visualização da mensagem cifrada utilizando o log do kernel

```
Mensagem cifrada no arquivo: CB54B48A88696D1D19FDFD0B1D1B7FD16F0
```

Figura 29 - Mensagem lida do arquivo

Leitura do arquivo

```
Mensagem lida: Ordem e progresso  
RET: 1
```

Figura 30 - Mensagem lida e decifrada do arquivo

```
Nov 26 14:56:49 pedroit-VirtualBox kernel: [ 112.084264] Result Data: 4f 72 64  
65 6d 20 65 20 70 72 6f 67 72 65 73 73 6f 83 82 81 cd ee ff ff 80 83 82 81 cd ee  
ff ff Ordem e progresso.....
```

Figura 31 - Visualização da mensagem decifrada utilizando o log do kernel

Conclusão

Com o experimento realizado foi possível compreender todos os passos para a implementação e testagem de uma chamada de sistema (syscall) em um kernel linux. Com isso, todos os envolvidos tem a capacidade de desenvolver uma syscall caso necessário.

Também foi possível se familiarizar ainda mais com a API criptográfica do linux e alguns de seus algoritmos de cifra, no caso do experimento AES em modo ECB.