PUC CAMPINAS
PONTIFÍCIA UNIVERSIDADE CATÓLICA

Sistemas Operacionais B

Projeto 2
Crypto System Calls

Breno Baldovinotti	|	RA: 14315311
Caroline Gerbaudo Nakazato	|	RA: 17164260
Marco Antônio de Nadai Filho	|	RA: 16245961
Nícolas Leonardo Külzer Kupka	|	RA: 16104325
Paulo Mangabeira Birocchi	|	RA: 16148363

27/11/2019

CEATEC – Centro de Ciências Exatas, Ambientais e de Tecnologia

1.Introdução

A criptografia é de extrema importância no mundo atual, fornecendo segurança e confiabilidade às constantes trocas de informações. No Linux, não é diferente. Sua API criptográfica é amplamente utilizada em outras porções de kernel, e é fundamental para o correto funcionamento do sistema operacional, pois permite uma comunicação correta e segura com outros dispositivos ou até mesmo entre os próprios mecanismos do SO.

Além disso, as chamadas de sistema são de suma importância para os processos em nível de usuário. É somente através delas que é possível solicitar a um processo a utilização de recursos e/ou serviços do Sistema Operacional, e são o único ponto de entrada de um programa para o Kernel do sistema.

Portanto, este projeto visa a análise e o entendimento das técnicas de implementação de uma chamada de sistema para o Linux que faz uso da API criptográfica presente no Kernel do mesmo. Também, busca familiarizar-se com os detalhes de implementação de uma chamada de sistema, seus mecanismos e suas dependências.

Para isto, foram desenvolvidas duas chamadas de sistemas criptográficas: *write_crypt* e *read_crypt*, capazes de, respectivamente, armazenar arquivos cifrados e decifrar arquivos para leitura através do algoritmo AES em modo ECB, o qual utiliza-se de uma chave primária definida no código fonte da chamada de sistema, a qual só pode ser alterada através de uma modificação no código fonte sucedida por uma recompilação do kernel.

Para o teste das syscalls, foi desenvolvido um programa em espaço de usuário que faz uso das mesmas, podendo fazer tanto uma leitura quanto uma escrita criptografadas e, em seguida, exibir os retornos ao usuário.

2. Detalhes do Projeto

As chamadas de sistema foram desenvolvidas na versão 4.4.190 do Kernel Linux. Ambas utilizam o algoritmo AES em modo EBC 128 e possuem a chave primária definida em seus códigos fonte.

2.1 Funcionamento do Algoritmo

Na implementação das syscall de *read* e *write* criptografados, foi levada em consideração a implementação do próprio Linux e suas chamadas de escrita e leitura dentro do sistema de arquivos voltada para a arquitetura *86x*. Neste processo, em suma, o Kernel recebe do espaço de usuário um buffer pela chamada "*syscall(NÚMERO, […])*", onde *NÚMERO* é uma referência à chamada desejada, sendo encontrado na tabela *linux-kernel-source/arch/x86/entry/syscalls/syscall_64.tbl* do Kernel.

Tanto nas chamadas *read* quanto para nas chamadas *write*, em espaço de Kernel, é preciso tratar o buffer recebido do espaço de usuário, a fim de salvar no arquivo os dados já criptografados de forma que, quando o usuário realiza a escrita/leitura criptografada, esta se dá de forma transparente. Para tal, foi necessário a implementação de um algoritmo criptográfico, neste caso, o AES 128 em modo ECB, além de, também, operações de padding do bloco para o correto funcionamento da cifragem.

No código fonte das chamadas de sistema, é necessário informar uma chave simétrica, a qual é utilizadas pelo algoritmo AES ECB para cifrar e decifrar os dados. O algoritmo de criptografia AES possui um tamanho de bloco fixo em 128 bits, mas apresenta inúmeras variantes. Neste projeto, foi utilizada a variante AES ECB 128, o qual apresenta uma chave de comprimento de 128 bits.

O algoritmo do AES ECB, na cifragem, utiliza-se da chave primária fornecidos no código fonte da syscall, gravando o dado cifrado correspondente ao original fornecidos na chamada. A etapa de cifragem é realizada com cada bloco de dados afim de se obter a palavra cifrada final.

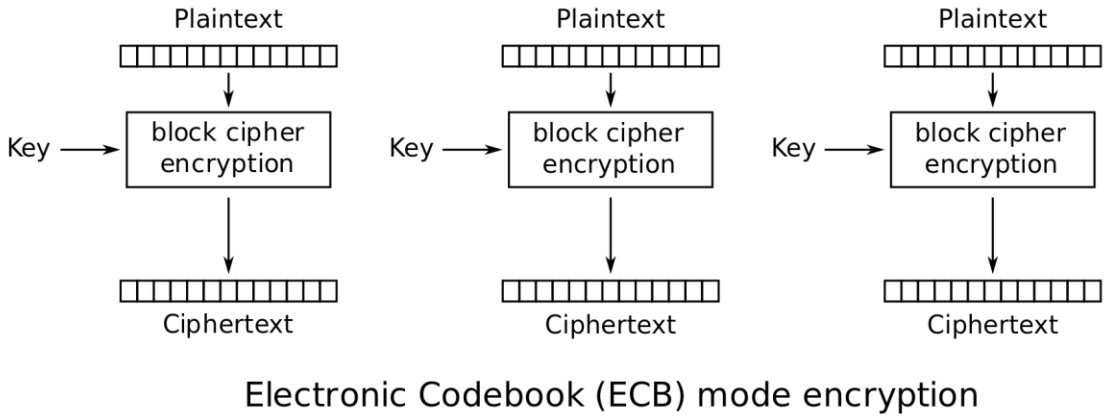

Figura 1: Diagrama do algoritmo de cifragem AES em modo ECB.

Já na decifragem, o processo inverso da cifragem é realizado, fornecendo como resposta o dado correspondente aos dados decifrados fornecidos na leitura do arquivo, utilizando-se também da mesma chave primária presentes no código fonte da chamada.

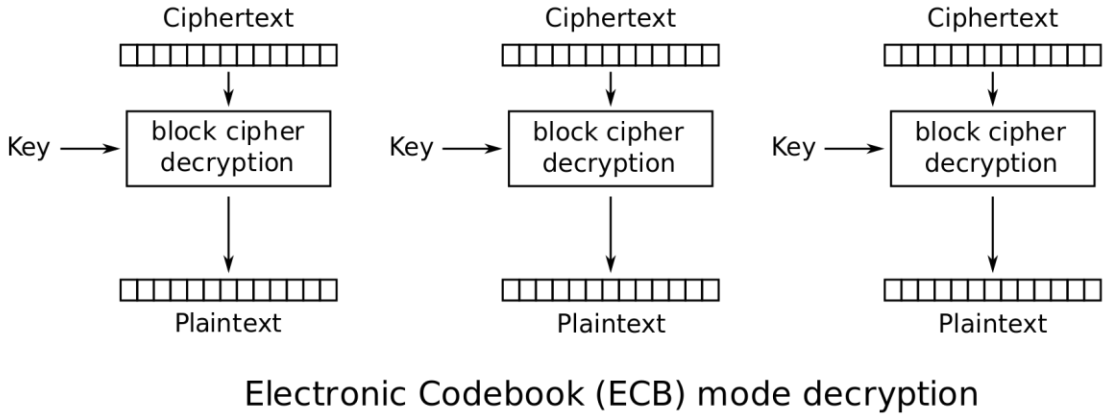

Figura 2: Diagrama do algoritmo de decifragrem AES em modo ECB.

Após a implementação do algoritmo de criptografia, iniciou-se a implementação das chamadas de sistema, onde o *write* consiste em criptografar, em espaço de Kernel, o buffer recebido pelo espaço de usuário, e, em seguida, enviar ao arquivo os bytes já criptografados. Já para o *read*, é enviado um buffer em espaço de usuário juntamente com o tamanho de dados a serem lidos do arquivo, e, então, são calculados quantos bytes devem ser lidos de acordo com o tamanho da chave primária (16 bytes), realizado pela escrita para obter os dados de forma coesa. Por fim, ao executar a rotina de decriptação, seu resultado é enviado para o buffer em espaço de usuário, tendo-se, assim, um processo transparente ao usuário.

2.2 Modificações no Código-Fonte do Kernel Linux

Primeiramente, em *linux-kernel-source* criou-se a pasta *rw_crypto* para os arquivos das chamadas de sistemas. Em seguida, em *arch/x86/entry/syscalls*, editou-se o arquivo "*syscall_64.tbl"* para incluir as novas chamadas de sistema na tabela de chamadas. Foi modificado este arquivo devido ao computador utilizado suportar e seu SO adotar a estrutura de 64 bits. Em seguida, foi editado o arquivo "*syscalls.h"*, localizado em *include/linux*, a fim de incluir as chamadas no header para serem utilizadas.

Com isso, foi utilizado o comando *make menuconfig* e carregado o arquivo *.config* enxuto gerado na Atividade 1 para recompilar o kernel com *make*, agora com as novas chamadas de sistema. Por fim, o novo kernel foi instalado com *make modules_install install.*

2.3 Programa em Espaço de Usuário

O programa de testes lê uma string digitada pelo usuário e acessa o arquivo "*out_teste.txt*", fazendo assim a execução da chamada *write_crypt* com o conteúdo dessa string, e, em, seguida, faz a chamada *read_crypt*.

Para os validar a cifragem e a decifragem realizadas, foi utilizado o site *http://aes.online-domain-tools.com/*, cujo algoritmo de padding é o mesmo utilizado no projeto, onde inserimos "0" a direita da palavra até que o bloco seja completo. Para os testes, foi comparado o output em hexa da *crypto write* através do comando *hexdump* com a saída em hexa do site. Para o *crypto read*, bastou verificar se o valor lido era o valor digitado originalmente.

3.Resultados

Primeiramente, criou-se a pasta *rw_crypto* em *linux-kernel-source,* onde foram colocados os arquivos "*crypto_ecb.c"*, "*crypto_ecb.h"*, "*read_write.c"* e o "*Makefile"*, os quais se encontram em anexo a este trabalho. A Figura 3 mostra os arquivos da pasta após a compilação e instalação do kernel customizado.

```
user@ubuntuvm:~/linux-4.4.190/rw_crypto$ ls
built-in.o    crypto_ecb.h  internal.h  modules.builtin  read_write.c
crypto_ecb.c  crypto_ecb.o  Makefile    modules.order    read_write.o
user@ubuntuvm:~/linux-4.4.190/rw_crypto$
```

Figura 3: Comando "ls" na pasta "linux-kernel-source/rw_crypto" após a compilação do Kernel.

Em seguida, em *arch/x86/entry/syscalls*, editou-se o arquivo "*syscall_64.tbl"* para incluir as novas chamadas de sistema na tabela de chamadas. Foi modificado este arquivo devido ao computador utilizado suportar e seu SO adotar a estrutura de 64 bits. Tais modificações podem ser observadas na Figura 4. Os números das chamadas *write_crypt* e *read_crypt* são, respectivamente, 546 e 547.

```
374     546     64      write_crypt             write_crypt
375     547     64      read_crypt              read_crypt
```

Figura 4: Inclusão das chamadas de sistema ao final do arquivo arch/x86/entry/syscalls/syscall_64.tbl.

Em seguida, em *include/linux*, foi editado o arquivo "*syscalls.h"* a fim de incluir as chamadas no header para serem utilizadas, como pode ser observado na Figura 5.

```
901   asmlinkage long write_crypt (int fd, void * buf, size_t count);
902   asmlinkage long read_crypt (int fd, void * buf, size_t count);
```

Figura 5: Inclusão das chamadas no header ao final do arquivo include/linux/syscalls.h..

Então, foi utilizado o comando *make menuconfig* e carregado o arquivo *.config* enxuto gerado na Atividade 1 para, então, recompilar o kernel através do comando *make*, agora com as novas chamadas de sistema inclusas. Por fim, o novo kernel foi instalado com *make modules_install install.*

```
user@ubuntuvm:~$ uname -r
4.4.190-marc0.proj2-v3
```

Figura 6: Versão Customizada do Kernel que incluí as novas Chamdas de Sistema.

Para os testes deste projeto, foi feito um programa em espaço de usuário, chamado de *teste.c*, o qual lê e escreve em um arquivo "*out_teste.txt"*. Trata-se de um programa que executa primeiro a chamada write_crypt e depois a chamada read_crypt para testes. Tal execução pode ser vista na Figura 7.

```
user@ubuntuvm:~$ ./teste
Entre com uma string a ser criptografada:
teste123
String READ=teste123
string_read[0]=0X74
string_read[1]=0X65
string_read[2]=0X73
string_read[3]=0X74
string_read[4]=0X65
string_read[5]=0X31
string_read[6]=0X32
string_read[7]=0X33
user@ubuntuvm:~$
```

Figura 7: Execução do programa de testes em modo de escrita, com entrada "teste123".

A string "*teste123"* foi escolhida arbitrariamente. Para validar a criptografia, fez-se, no terminal, o comando *cat out_teste.txt* a fim de verificar se era possível ler as informações do arquivo.

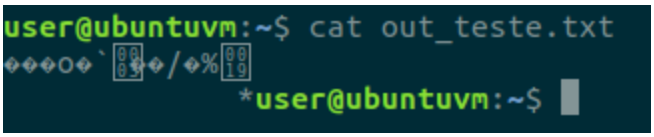

Figura 8: Output do comando "cat" no arquivo "out_teste.txt"

Com isso, percebeu-se que o arquivo não poderia ser lido normalmente. A fim de validar o correto funcionamento da criptografia, comparou-se o resultado do *hexdump* do arquivo com o resultado em hexadecimal do site *AES Online Domain Tools.*

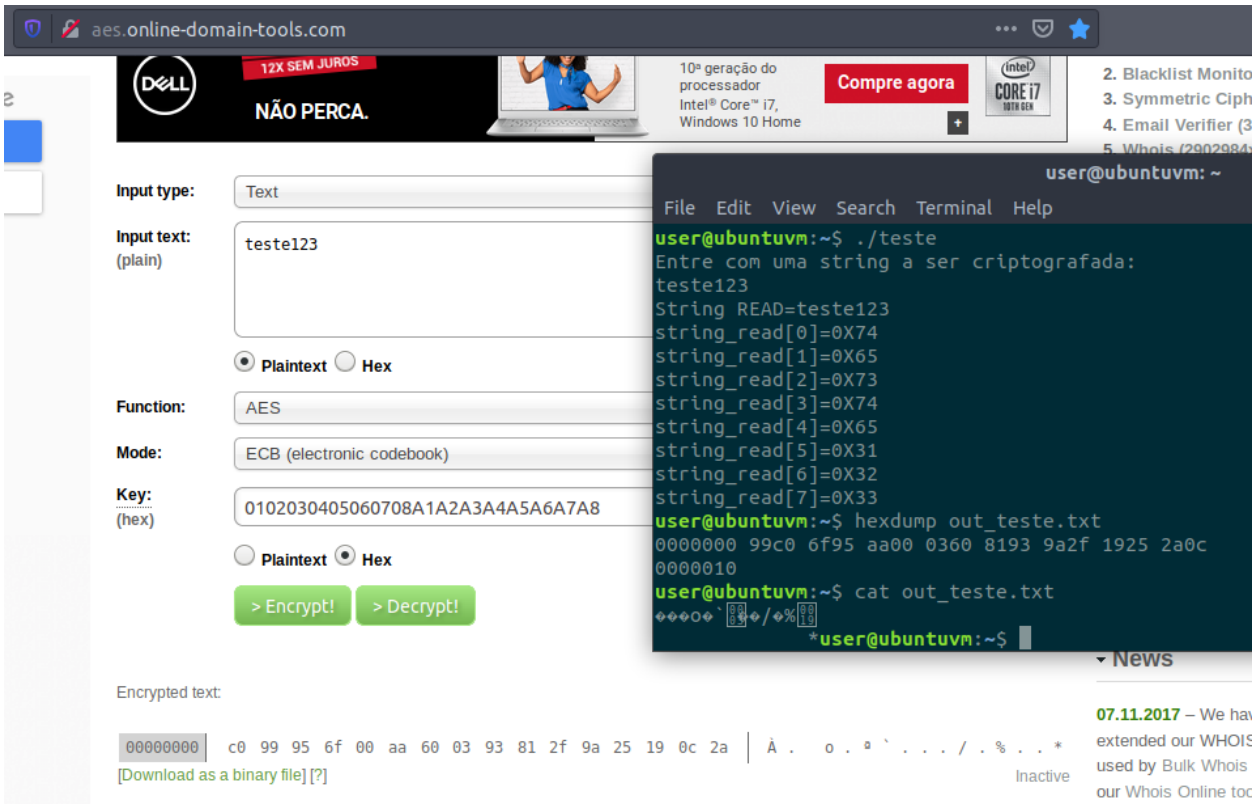

Figura 9: Comparação de resultados de cifragem no site "http://aes.online-domain-tools.com/".

É possível observar uma diferença entre a exibição do site e a do comando *hexdump*. Isso se deve a maneira na qual os dados são armazenados dentro do arquivo. Contudo, foi realizado um debug através do comando *dmesg*, a fim de se provar a consistência do resultado da criptografia.

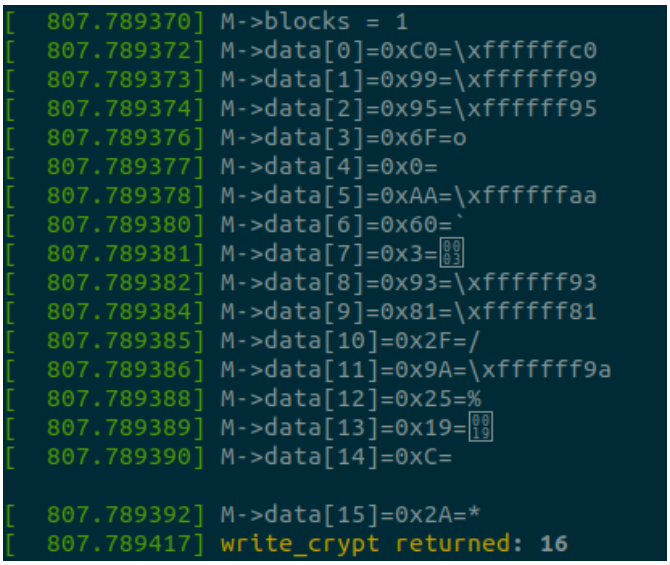

Figura 10: "dmesg" do resultado da criptografia.

Com isso, foi verificada a correta criptografia do dado pela chamada write_crypt.

Para o teste da leitura, como já foi confirmado pelo *cat out_teste.txt* que uma simples leitura não é efetiva, apenas fez-se a execução do programa de testes, obtendo-se o resultado "*String READ=teste123*", como pode-se ver na Figura 7. Esta pode ser vista em mais detalhes através do comando *dmesg*.

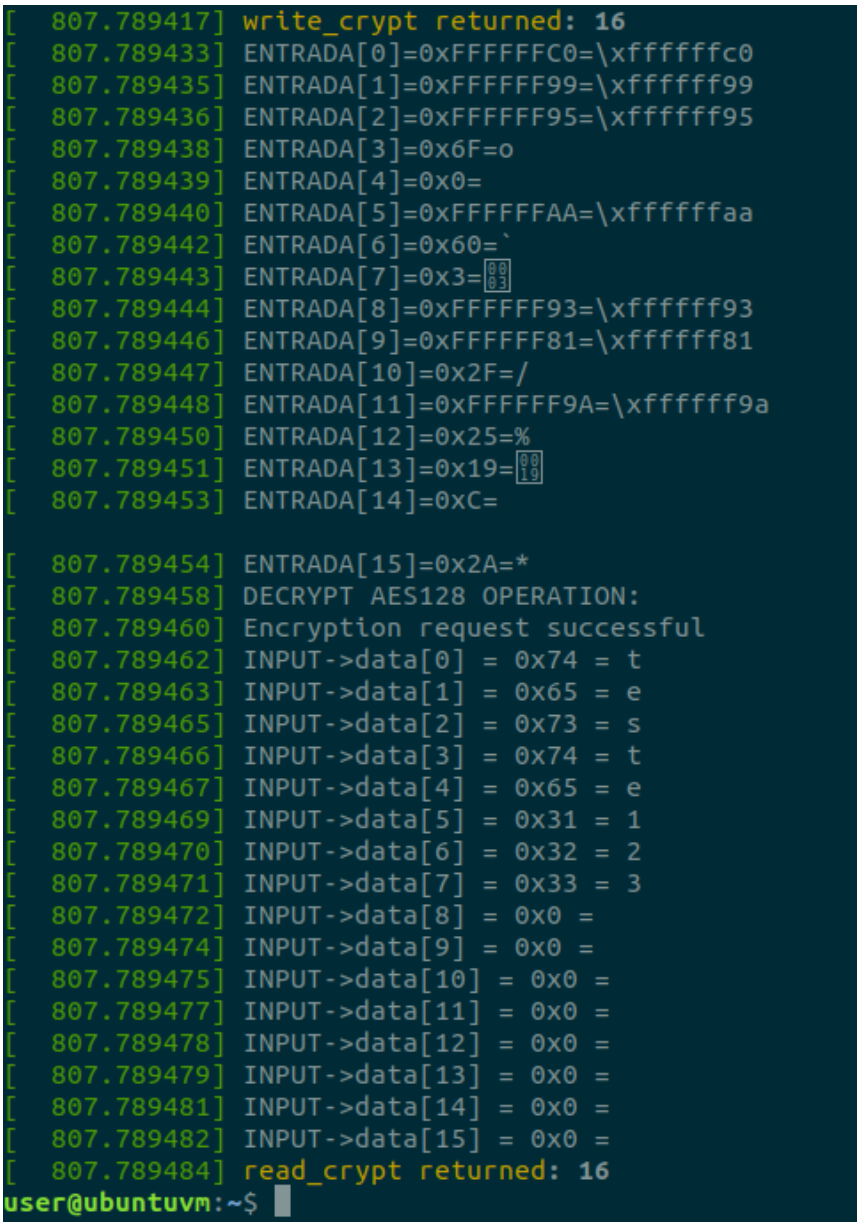

Figura 11: "dmesg" da descriptografia e read_crypt.

Com isso, foi verificado o correto funcionamento da leitura dos dados criptografados através da chamada *read_crypt*.

4.Conclusão

O projeto funcionou como esperado, apresentando o correto funcionamento das chamadas de sistema.

Contudo, na etapa final do prazo de conclusão do projeto, deu-se conta de uma falha operacional que ocorre quando o processo de escrita é setorizado em vários *writes* para os casos em que a quantidade de bytes não é um múltiplo de 16. Por exemplo, ao realizar 3 *writes* de 1 byte, gera-se um arquivo de 48 bytes (3 x 16 bytes). Entretanto, ao executar-se um read de 3 bytes, é obtido apenas o resultado do primeiro bloco, contendo seus 3 primeiros bytes. Isto é: o primeiro byte descriptografado seguidos de 0x00, 0x00, fruto do padding executado, deixando-se, assim, de receber o conteúdo dos outros 2 bytes restantes escritos. Portanto, reconhece-se que foi implementar uma solução ótima para resolver tal problema dentro do tempo esperado. Entretanto, elaborou-se uma solução primordial, que consiste em, para cada *write* executado, armazena-se no arquivo, antes dos dados, um registro, chamado de *burst_count*, de quantos bytes foram escritos naquele *write*. Com esta informação armazenada no arquivo, mesmo que ocorram *writes* subsequentes, basta modificar a chamada *read* para interpretar este registro de forma a manter a consistência dos dados. Uma possível solução consiste em um algoritmo executado pelo *read*, o qual verificaria o *burst_count* de cada *write* e registraria somente os bytes que interessam, ignorando o padding, em um registrador temporário em loop, até que todos os bytes desejados sejam coletados, e, assim, passando para o espaço de usuário os dados consistentes. Outro fator que poderia ser melhorado seria a chamada das funções por nome, ao invés de seus números na tabela de syscalls.

Contudo, Através deste projeto, foi possível familiarizar-se com os detalhes de implementação de chamadas de sistema ao Kernel Linux e aprimorar os conhecimentos acerca de sua API criptográfica. Através do estudo destas chamadas de sistema, foram aprimorados os conceitos de implementação, adição, modificação e testes de chamadas de sistemas no kernel que seja capaz de realizar escritas cifradas e leitura de dados decifrados.