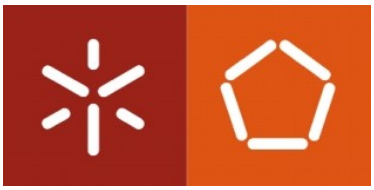

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Segurança de Sistemas Informáticos

Autorização de Operações ao nível do sistema de ficheiros

- Trabalho Prático 3 -

Filipe de Sousa Marques

Laércio Sartori Andrade Junior

16 de Janeiro de 2020

Conteúdo

1 Introdução

Este trabalho foi desenvolvido no âmbito da Unidade Curricular de Segurança de Sistemas Informáticos, pertencente ao plano de estudos do 4º ano do Mestrado em Engenharia Informática.

Com este trabalho pretende-se complementar os mecanismos de controlo de acesso de um sistema de ficheiros tradicional do sistema operativo Linux com um mecanismo adicional de autorização de abertura de ficheiros. O mecanismo desenvolvido, trabalhará a partir do terminal, onde sempre que um utilizador tente ler um ficheiro, terá que se autenticar por meio de sms. Para tal existe um pequeno banco de dados que associa o `user id` ao seu número de telemóvel, para o qual será enviado um código de validação que terá de ser inserido na consola. Este sistema foi concretizado com recurso a uma API externa para o envio de sms's e o sistema de ficheiros 'protegido' tem como base a biblioteca `libfuse`.

2 Guia execução

1. Definir permissões e compilar código .c

   ```
   $$ cd /root/fuseAuth
   $$ chmod 600 ruby_script.rb
   $$ make
   ```

2. Criar pasta temporária para o novo sistema de ficheiros

   ```
   $$ mkdir /tmp/pt_dir
   ```

3. Montar sistema de ficheiros libfuse na pasta criada

   ```
   $$ sudo ./passthrough -o allow_other -o default_permissions /tmp/pt_dir
   ```

4. Ir para a nova unidade montada, e **ler** um ficheiro qualquer de texto exemplo com `cat`

   ```
   $$ cd /tmp/pt_dir
   $$ cat example.txt
   ```

5. Deverá ser enviado um sms com código de autenticação de 6 dígitos

6. Abrir novo terminal, ir para pasta onde se tem o código compilado e enviar os seis dígitos ao sistema de ficheiros pelo *fifo* (solução provisória)

   ```
   $$ cd /root/fuseAuth
   $$ echo -n "123456" > get_code
   ```

7. O sistema de ficheiros dará autorização para ver o conteúdo

8. Se der erro, ou aparecerá na consola "Permission denied"ou bloqueará na abertura do ficheiro. Neste caso poderá ser necessário fazer reboot à máquina virtual, aguardar um pouco, e recomeçar o processo.

3 Adicionar novos *users*

Para adicionar novos utilizadores terá de se ter permissões administrativas para editar o ficheiro `ruby_script.rb` e adicionar uma entrada com mapeamento `uid : telemóvel` à variável global *$users*. Pode-se obter o `uid` de determinado *user* a partir do terminal, estando o *user* autenticado no sistema,

```
$$ id -u
> 501
```

4 Implementação

4.1 Abertura ficheiros

Para a nossa implementação aproveitamos o que estava já feito do ficheiro exemplo `passthrough.c`. Esta implementação exemplo faz um *'mirror'* da hierarquia de ficheiros existente da *root* do sistema, e faz o re-encaminhamento de todos os pedidos para acesso a este *'mirror'* para as *syscalls* default do sistema.

A função no `passthrough.c` que faz o re-encaminhamento para abertura de ficheiros é a `xmp_open()`.

Esta função procura o descritor do ficheiro presente em `const char *path` com a *syscall* `open` e devolve-o ao sistema de ficheiros em caso de sucesso, isto é, se o ficheiro existe e o *user* tem permissões para ler o mesmo.

A estratégia será

1. Antes de enviar o descritor devolvido por `open`, gerar um código aleatório a enviar por sms ao utilizador que fez o pedido de leitura
2. Aguardar até que o código seja inserido, e , após validado o código, libertar o descritor do fichero para o sistema de ficheiros
3. Caso o código esteja errado, devolver mensagem de erro apropriada

4.2 Enviar código por sms

Dentro da função `xmp_open()` após verificarmos que realmente o *user* terá permissão para levar avante a abertura do ficheiro, chamamos a função `create_token()`. O funcionamento de `create_token()` será

```
int create_token(char* request_id) {
    // para guardar token
    char buffer[100];

    mkfifo('get_code'); // fifo para ler código sms enviado pelo cliente
    mkfifo('get_request_id'); // fifo para ler token enviado pela api externa

    user_id = fuse_get_context()->uid;

    // num processo em separado, enviar sms ao user que fez pedido abertura ficheiro
    system('ruby ruby_script send_sms %user_id%'); // retorna um request_id

    // ler token request_id do fifo 'get_request_id' enviado pela api externa
    while(read('get_request_id', &buffer[i], 1) > 0)
        i++;

    // retorna token request_id
```

```
    strcpy(request_id, buffer);
}
```

Para o envio dos sms's usamos uma API externa *'nexmo'*. As suas configurações são feitas num *script* externo `ruby_script.rb`, assim como o mapeamento `user ids : números telemóvel`.

O envio de um sms pode ser despoletado manualmente fazendo na consola

```
ruby ruby_script.rb send <uid>
```

4.3 Validar código inserido

Para validação do código, teremos que aguardar que o *user* receba o código no telemóvel, insira-o na consola, e seja validado pela API externa e só depois a permissão será dada de leitura. A função que trata este processo é a `validate_code()`

```
int validade_code(char *request_id) {

    // bloqueia à espera que o user envie o código sms para o fifo get_code
    open('get_code');

    // após lido o código 6 dígitos, valida o código pela API externa
    result = system('ruby ruby_script.rb verify %code %request_id_token');

    if(result)
        return SUCCESS;
}
```

5 Vulnerabilidades

Uma das vulnerabilidades identificadas neste sistema, que usa `fifos`, é que estes ficheiros usados para comunicação entre processos são visíveis a outros utilizadores com permissões administrativas, pelo que um *user* que seja *root*, tem possibilidade de interceptar os conteúdos dos mesmos. Uma solução seria fazer uma função de encriptação do lado do cliente que envia o código sms, e desencriptação do lado do servidor que valida o código.

6 Conclusões e sugestões de melhoria

Esta última fase do trabalho permitiu-nos sintetizar o conhecimento adquirido nas aulas sobre `unix` e controlo de permissões. Além disso aprendemos um pouco mais sobre sistemas de ficheiros e permitiu repescar conhecimentos adquiridos noutras UCs relativas a sistemas operativos.

Ao longo do desenvolvimento desta solução tivemos dificuldades com a instalação da biblioteca `libfuse`, que apesar da existência de guias online, nos deparamos com vários erros e bugs onde tivemos que procurar soluções *ad-hoc*. Para o caso da edição do `passthrough.c` a documentação é inexistente.

Pensamos os objectivos deste trabalho terem sido alcançados, no entanto apontamos como faltas da nossa implementação o facto de não termos uma interface gráfica para validar os códigos enviados por sms, e a falta de implementação de um *timeout* que seria conseguido à custa de sinais em `.c`.

Posto isto, como melhorias futuras para este sistema, pensamos que seria interessante investir também nas seguintes funcionalidades

- Permitir a utilização de vários *users* em simultâneo, que poderia ser conseguida adicionando mais *'fifos'* de comunicação, específicos para cada utilizador
- Permitir abertura de mais ficheiros consecutivamente, mais rapidamente. De momento o sistema de sms's adoptado só enviará no máximo uma mensagem por minuto. Uma solução para isto poderá ser o seguinte ponto,
- Guardar durante x tempo o código de validação enviado por sms, de forma a não pedir ao utilizador verificações futuras

7 Referências

https://github.com/libfuse/libfuse

https://github.com/torvalds/linux/blob/
a33f32244d8550da8b4a26e277ce07d5c6d158b5/
Documentation/filesystems/fuse.txt

https://libfuse.github.io/doxygen/passthrough_8c.html

https://man.openbsd.org/fuse_get_context.3

https://opensource.com/article/19/4/interprocess-communication-linux-channels

https://www.cyberciti.biz/faq/create-a-user-account-on-ubuntu-linux/

https://developer.nexmo.com/verify/overview

https://developer.nexmo.com/api/verify

A Anexo: Ficheiro API externa em Ruby

```
require 'bundler/inline'

gemfile do
  source 'https://rubygems.org'
  gem 'nexmo'
end

## map uid's to phone numbers - sensitive data, please secure this file
## to find your uid type 'id -u' in console
$users = {
:'0' => '351000000000'
}

$client = Nexmo::Client.new(
  api_key: '073d7b17',
  api_secret: '0aSE2jrehK6KDsh7'
)

class SmsService
  def self.send(uid)

    phone_number = $users[uid.to_sym].nil? ? 'unknown' : $users[uid.to_sym].nil?

    response = $client.verify.request(
      number: phone_number,
            country: 'PT',
            brand: 'FILE AUTH',
            code_length: '6',
            pin_expiry: '120'
    )

    log_output = open('/root/fuseAuth/auth.log','a+')
```

```
  log_output.puts 'uid: ' + uid + ' attempted sent message to ' + phone_number
  log_output.flush

  if response.status == '0'
    request_id = response.request_id
    puts request_id
    log_output.puts 'uid: ' + uid + ' ' + request_id
    log_output.flush
    log_output.close
    output = open('/root/fuseAuth/get_request_id','w');
    output.puts request_id
    output.flush
    output.close
    exit(0)
  else
    puts response.error_text
    log_output.puts 'uid: ' + uid + ' ' + response.error_text
    log_output.flush
    log_output.close
    output = open('/root/fuseAuth/get_request_id','w');
    output.puts "-1" ## send error signal
    output.flush
    output.close
    exit(1)
  end
end

def self.verify(code, request_id)
  response = $client.verify.check(
    request_id: request_id,
    code: code
  )

  output = open('/root/fuseAuth/auth.log','a+');
```

```
    if response.status == '0'
      # the cost of this verification
      # puts 'Cost ' + response.price + response.currency
      puts response.status
      output.puts  'SUCCESS Code ' + code + ' : ' + request_id #log
      output.flush
      output.close
      exit(0)
    else
      output.puts response.error_text #log
      output.flush
      output.close
      puts response.error_text
      exit(1)
    end
  end
end

if ARGV[0] == "send"
  SmsService.send(ARGV[1])
else
  SmsService.verify(ARGV[1], ARGV[2])
end
```

B Anexo: Função .c passthrough

```
static int xmp_open(const char *path, struct fuse_file_info *fi)
{
    int res;
    char *request_id = (char*) malloc(sizeof(char)*200);
    int size = 0;

    res = open(path, fi->flags);
    if (res == -1)
        return -errno;

    if(create_token(request_id, &size) == 1) //send sms
        return -1;

    //validade code
    if (strncmp(request_id, "-1", 2) != 0 && validate_code(request_id) == 0)
    {
        //send file handle id to user
        fi->fh = res;
        return 0;
    }
    else //invalid code
    {
        close(res);
        return -13;
    }
}
```

C Anexo: Funções .h passthrough_helpers

```
unsigned create_token(char* request_id, int* size)
{
char command[200];
const char* file_code = "/root/fuseAuth/get_code";
mkfifo(file_code, 0640);
const char* file_request_id = "/root/fuseAuth/get_request_id";
mkfifo(file_request_id, 0640);

uid_t user_id = fuse_get_context()->uid;
sprintf(command, "ruby /root/fuseAuth/ruby_script.rb send %u", (unsigned) user_id);

// send sms
pid_t pid = fork();
if(pid == 0)
{
// Substituir o filho pelo programa que queremos executar
system(command);
exit(0);
} else {
// if sms sent successfully, get returned 'request_id'
if(1)
{
char buf[100];
int fd_request_id = open(file_request_id, O_RDONLY);

if (fd_request_id == -1)
return -errno;

int i = 0; int count;
while ((count = read(fd_request_id, &buf[i], 1)) > 0)
i++;
```

```
buf[i] = '\0';
close(fd_request_id);

strncpy(request_id, buf, i);
*size = i;

return 0;
}
else
{
return 1;
}
}

}

unsigned validate_code(char* request_id)
{
char buf[100];
char command[200];
const char* file = "/root/fuseAuth/get_code";

int fd = open(file, O_RDONLY);

if (fd == -1)
return -errno;

int i = 0; int count;
while ((count = read(fd, &buf[i], 1)) > 0)
i++;

buf[i] = '\0';
close(fd);
```

```
sprintf(command, "ruby /root/fuseAuth/ruby_script.rb verify %s %s", buf, request_id);
int returned_code = system(command);

if(returned_code == 0) //code is correct
{
return 0;
}
else //wrong code
{
return 1;
}

}
```