

UNIVERSIDADE DO MINHO

MESTRADO INTEGRADO EM ENGENHARIA

INFORMÁTICA

Trabalho Prático 3

Diogo Pinto Ribeiro, A84442

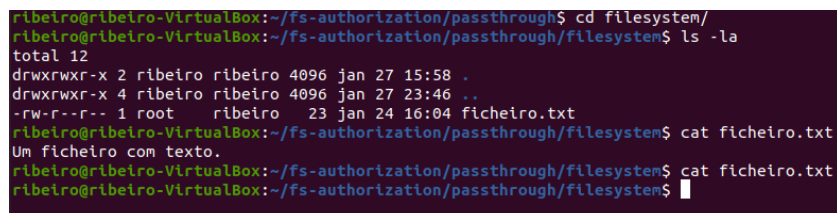
Luís Pedro Barbosa Ferreira, A86265

Segurança de Sistemas Informáticos
4º Ano, 1º Semestre
Departamento de Informática

7 de fevereiro de 2021

1 Introdução

Para a Unidade Curricular de Segurança de Sistemas Informáticos, foi proposta a adição aos mecanismos de controlo de acesso de um sistema de ficheiros tradicional do sistema operativo Linux, um mecanismo adicional de autorização de operações de abertura de ficheiros. O mecanismo desenvolvido foi concretizado com base num novo sistema de ficheiros baseado em libfuse. A nossa implementação passa por duas componentes como iremos ver de seguida: a componente libfuse que contém o sistema de ficheiros e efetua pedidos de autorização quando necessário, e, um servidor de autorização que responde a estes pedidos e permite a gestão de autorizações.



```
ribeiro@ribeiro-VirtualBox:~/fs-authorization/passthrough$ cd filesystem/
ribeiro@ribeiro-VirtualBox:~/fs-authorization/passthrough/filesystem$ ls -la
total 12
drwxrwxr-x 2 ribeiro ribeiro 4096 jan 27 15:58 .
drwxrwxr-x 4 ribeiro ribeiro 4096 jan 27 23:46 ..
-rw-r--r-- 1 root    ribeiro  23 jan 24 16:04 ficheiro.txt
ribeiro@ribeiro-VirtualBox:~/fs-authorization/passthrough/filesystem$ cat ficheiro.txt
Um ficheiro com texto.
ribeiro@ribeiro-VirtualBox:~/fs-authorization/passthrough/filesystem$ cat ficheiro.txt
ribeiro@ribeiro-VirtualBox:~/fs-authorization/passthrough/filesystem$
```

Na figura apresentada encontra-se o exemplo de dois acessos: o primeiro em que o utilizador insere a password e acede ao conteúdo, e o segundo, onde não é dada a autorização dentro do tempo esperado e não é dado o acesso.

2 Arquitetura

2.1 Sistema de Ficheiros

O sistema de acesso a ficheiros foi implementado com a ajuda de FUSE. Este é uma interface para **UNIX** que serve de ponte para os verdadeiros interfaces do kernel e permite que utilizadores executem certas operações que de outra maneira não teriam permissões para o fazer.

2.1.1 Ferramentas e Bibliotecas

Com o objetivo de tornar possível a implementação de um *filesystem* num programa em *userspace* foram integradas, no nosso trabalho, as seguintes bibliotecas:

fuse3, **libfuse3** e **libfuse3_dev**. Foi criada uma **makefile** para só ser preciso correr o comando **make**, quando se quiser compilar e correr o programa e **make umount**, para fazer *umount* do *filesystem*.

Ainda como apoio à implementação deste *software* foi usado um *template* que já fornecia código base para fazer o *wrap* a uma lista vasta de *system calls* do *linux*, como, **open**, **write**, **mkdir**,...

Todo o código relacionado com o *fuse* pode ser encontrado nos ficheiros **passthrough.c** e **passthrough_helpers.h** e são nestes onde ocorrem as comunicações com o servidor, que será falado mais à frente. Esta comunicação é realizada através da ferramenta **CURL** utilizada para obter ou enviar dados usando a sintaxe **URL**.

2.1.2 Funcionamento

Para demonstrar o funcionamento do nosso *filesystem* foi criada uma pasta chamada **actual**, onde se vai situar o sistema que vamos querer replicar/simular. É a pasta onde o sistema fica montado e replicado recebe o nome de *filesystem*.

Sempre que o utilizador se encontra no nosso *filesystem* e realiza alguma ação que recorre a uma *system call*, o programa criado é evocado, fazendo o papel de um *wrapper*. Este vai começar por verificar se o utilizador está autorizado a realizar a ação desejada, recorrendo à função **fuse_get_context()** fornecida pela biblioteca *fuse*. Através desta é obtido o id do utilizador e, consequentemente, o seu nome. É, então, também adquirida a informação do dono do ficheiro que está a sofrer a ação, para verificar se é o mesmo utilizador que está a executar a operação, e toda a atividade é guardada num ficheiro de logs no caminho **/var/log**.

Em caso positivo, não é necessário o programa realizar nenhum procedimento especial, logo é executada a *system call* normalmente. Em caso negativo, já é necessária intervenção do nosso programa, que vai enviar um pedido **HTTP POST** ao servidor através da ferramenta **CURL**. Esta mensagem vai enviar toda a informação necessária ao servidor, nomeadamente o utilizador que se encontra a executar a operação, o dono do ficheiro alvo, o nome da operação e o nome do ficheiro alvo.

Caso o servidor esteja operacional será recebido um código, usado para verificar

se a operação já foi permitida ou não. Caso haja algum problema no servidor e não seja recebida a resposta desejada, a operação será negada, de imediato, ao utilizador.

Seguindo pela situação que tudo correu bem, é utilizado o código recebido para de 5 em 5 segundos mandar pedidos `HTTP GET` durante 30 segundos ou até ser recebida uma resposta positiva de permissão ao utilizador. Se for recebida uma resposta positiva é, finalmente, efetuada a operação como se do dono se tratasse. É, ainda, adicionada num ficheiro de logs a resposta final proveniente do servidor.

Houve um cuidado especial nas *system calls* `create` e `mkdir`, onde foi necessário executar um `chown` para mudar o dono e o grupo para os do utilizador a executar a operação, pois quando eram criados novos ficheiros ou diretorias, a root ficava como dono e grupo por como defeito, por ser este quem executa o nosso programa. O acesso a uma operação é avaliado com a função `isAuthorized`, que está implementada nas operações `access`, `mkdir/rmdir`, `unlink`, `chmod`, `chown`, `create`, `open`, `read` e `write`.

2.2 Servidor de Autorização

O servidor de autorização foi construído em **Node.js** e tem como principal função receber pedidos do sistema de ficheiros e efetuar a gestão das autorizações. Para esse efeito é disponibilizado um conjunto de rotas, e, é necessária a configuração inicial deste mesmo servidor, para que seja capaz de contactar todos os utilizadores. A principal vantagem desta implementação consiste no facto de não ser necessário correr o servidor na máquina que contém o sistema de ficheiros. É possível corrê-lo na mesma máquina, tal como numa outra máquina qualquer. Para o armazenamento de dados foi utilizado o **MongoDB**, dado que não serão armazenadas relações, e dado que o número de pedidos será de elevado número.

O funcionamento deste servidor passa por ser capaz de receber pedidos por parte do sistema de ficheiros. Estes pedidos necessitam de ter no seu corpo os seguintes campos: **user** (o utilizador que requisita autorização), **owner** (o utilizador que irá fornecer autorização), **operation** (a operação que se pretende efetuar) e o **target** (o objeto sobre qual a operação incide). Ao receber um **POST** com estes campos,

o servidor gera 2 códigos com caracteres hexadecimais começados por **SSI**, sendo que um dos códigos é enviado por email para o **owner** (utilizando o seu contacto definido no ficheiro config.js) para efeitos de autorização, e o outro código serve para efeitos de consulta de estado. O código de autorização é secreto e não deve ser divulgado, sendo que o código de consulta pode ser difundido sem restrições. Todos os pedidos efetuados são guardados numa coleção da base de dados definida no MongoDB.

O servidor possui duas funcionalidades além da mencionada. Com o código de consulta mencionado, é possível inquirir o servidor acerca do estado de um pedido, isto é, se o **owner** já inseriu o código na plataforma. Em caso positivo é retornado a string **true**, e em caso contrário é retornado a string **false**. Com o código de autorização é possível inseri-lo na página que contém o formulário, atualizando o estado do pedido em causa para **autorizado**.

As rotas disponibilizadas pelo servidor são as seguintes:

```
GET /  
  
// Rota para aceder o estado de um pedido  
GET /access/:id  
  
//Rota para criar um pedido  
POST /access
```

E o corpo de um POST para a criação de um pedido tem os campos **user**, **owner**, **operation** e **target**.

2.2.1 Instalação e Utilização

Para se proceder à utilização deste servidor são necessários alguns pré-requisitos, nomeadamente possuir o **mongo**, **node** e **npm** instalados na máquina.

Após estes requisitos estarem prontos, basta navegar até à pasta onde se encontra o servidor de autorização e executar os seguintes comandos:

```
npm i & npm start
```

Todas as dependências necessárias serão instaladas e o servidor estará à escuta na **porta 3000**. Caso seja necessário alterar ou adicionar utilizadores e respetivos endereços eletrónicos, basta acrescentar/alterar as respetivas entradas. Para efetuar a operação são necessários privilégios **root**.

3 Segurança

3.1 config.js

O primeiro grande problema que detetámos passa pelo ficheiro de configuração. Se um atacante quiser contornar o sistema, basta alterar o email do **owner** para o seu próprio endereço e torna todo o sistema inútil. Para mitigar isso, corremos os seguintes comandos (colocar o root como dono, e revogar as permissões de escrita dos restantes) para que **apenas o administrador (root) seja capaz de efetuar alterações**:

```
sudo chown root config.js
sudo chmod o-w config.js
sudo chmod g-w config.js
```

3.2 Vulnerabilidades

A nível de vulnerabilidades do software utilizado, foi efetuada uma pesquisa para perceber a que tipo de vulnerabilidades este está exposto. Relativamente ao MongoDB, a página oficial do MongoDB inclui uma avaliação de vulnerabilidades existentes, sendo que usando a versão mais recente (que é o nosso caso) não existem quaisquer tipos de vulnerabilidades conhecidas, relativamente a este.

As vulnerabilidades conhecidas em módulos existente no **npm** podem facilmente ser auditadas utilizando o comando **npm i**. Ao correr este comando, além de serem instaladas dependências em falta, são auditadas todas as que se encontram instaladas e listadas as vulnerabilidades encontradas. No nosso caso concreto foram sinalizadas **3 vulnerabilidades de severidade baixa**. Correndo o comando **npm audit fix --force** fomos capazes de mitigar todas as vulnerabilidades existentes. Relativamente ao **libfuse** encontrámos uma vulnerabilidade ativa que ainda

não foi corrigida e remonta a 2006 quando está ativa a opção *allow_other*. Se não for usada a opção *default_permissions* ao montar o sistema, o resultado da primeira verificação de permissões feita pelo sistema de ficheiros para uma diretoria será usado para acessos subsequentes enquanto o inode da diretoria acedida estiver presente na cache do kernel.

3.3 Controlo de Acesso no MongoDB

Ainda sobre o MongoDB, outro aspeto fundamental para assegurar a integridade dos dados e a segurança da informação é o controlo de acesso à base de dados. Para esse efeito, o MongoDB possibilita o acesso através de **utilizadores, sendo incluídos um username e password em todos os pedidos/operações efetuados sobre os dados**. Para uma versão de produção seria aconselhado implementar isto, no entanto, como estamos num contexto académico e com o intuito de tornar mais fácil a configuração em contexto de avaliação, não ativamos a opção no MongoDB. Apesar de não termos este requisito ativo, percebemos a **necessidade de controlar o acesso aos dados**.

3.4 Validação de Input

Outro aspeto que tivemos em conta foi a validação de input. Tendo alguns cuidados neste aspeto, podemos vir a prevenir potenciais casos de **injeção de comandos**, de **buffer overflows** ou mesmo de tentativas de **Denial of Service**.

A primeira validação que efetuámos é nos dados recebidos na operação **POST**. Verificámos se todos os campos **são compostos exclusivamente por caracteres alfanuméricos (e mais alguns caracteres especiais)** e excluímos todos os POST's que não cumprem o requisito. Outra validação que efetuamos é relativa ao **comprimento dos valores**, isto é, os campos **owner, user e operation devem ter um comprimento inferior a 32 caracteres** e o campo **target inferior a 255 caracteres**. Estes valores foram definidos após termos pesquisado acerca de limites de tamanho de user's e de caminhos em UNIX. A mesma validação foi feita nos pedidos de consulta e de autorização. Deste modo conseguimos tornar o nosso sistema mais seguro e prevenir potenciais problemas de segurança no futuro.