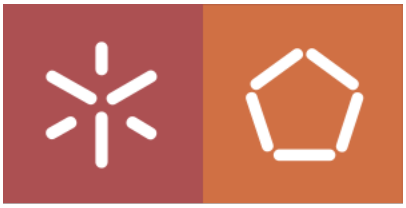

Universidade do Minho
Escola de Engenharia

Autorização de Operações ao nível do Sistema de Ficheiros

Segurança de Sistemas de Informáticos

Trabalho Prático nº03

Grupo 12

[Alexandre Ferreira nºA84961, Miguel Cardoso nºA85315]

1 Descrição da Solução

1.1 Filesystem

Inicialmente, e de forma a nos ambientarmos à *libfuse*, começamos por testar a implementação do *filesystem* ***passthrough*** nos exemplos da biblioteca. Utilizando este exemplo como base conseguimos perceber o funcionamento da biblioteca e do próprio sistema de ficheiros criado. No entanto, e com o avançar do desenvolvimento do projeto, decidimos manter esta implementação como base para o nosso *filesystem*, espelhando assim o sistema de ficheiros em **"/"**, ou seja, na raiz do sistema de ficheiros do SO.

Para facilitar a organização do código e limitar a quantidade de funcionalidades que um *user* poderá ter no nosso *filesystem*, retiramos algumas delas, comparativamente ao exemplo *passthrough* original, sendo que apenas ficaram operações como: *xmp_readdir, xmp_mkdir, xmp_rmdir, xmp_chmod, xmp_chown, xmp_open, xmp_create, xmp_read, xmp_write, xmp_getattr*, entre outras. Estas operações foram escolhidas por serem as funcionalidades mais comuns num sistema de ficheiros, e as que um utilizador mais necessita.

Na operação ***xmp_open*** é onde todo o controlo de permissões implementado entra em ação. Tal como será explicado mais à frente, existe um conjunto de mecanismos que permitem, ou não, desencadear a autorização pelo proprietário do ficheiro a ser acedido.

No início do desenvolvimento do projeto estivemos com algumas dúvidas referentes às permissões e quem poderia entrar no sistema de ficheiros, já que apenas o *root* e o *user* criador do *filesystem* tinham acesso ao mesmo. No entanto, e à medida que aprofundávamos o conhecimento do FUSE e *libfuse*, percebemos que poderíamos definir que qualquer utilizador poderia ter acesso, recorrendo à *flag* **"-o allow_other"**. Deste modo, qualquer utilizador poderia aceder ao sistema de ficheiros, contudo, numa primeira camada o sistema de permissões do linux atua e indica que permissões aquele utilizador tem (sendo dono terá permissões de dono, etc). Depois entra a camada desenvolvida neste projeto, que interseta a *system call* chamada e aplica a função correspondente definida. Neste local é verificado qual o utilizador a executar a *system call*, recorrendo à função ***fuse_get_context***, que preenche a ***struct fuse_context***, de onde retiramos o *uid* desse utilizador.

De seguida necessitamos de saber quem é o dono do ficheiro ao qual estão a tentar aceder. Conseguimos saber esta informação utilizando a função ***stat***, que recebe o *path* do ficheiro e, numa ***struct stat***, guarda as estatísticas desse ficheiro. Com esta *struct* conseguimos obter informação relevante como o uid do dono do ficheiro. No entanto, no registo dos utilizadores, não vamos fazer o mapeamento pelo uid do utilizador, mas sim pelo seu nome de utilizador, como tal, precisamos do saber qual o *username* que corresponde àquele uid. Conseguimos obter essa informação recorrendo à função ***getpwuid_r*** que utiliza o ficheiro *passwd* para preencher a ***struct passwd*** com informações do nome do utilizador, uid, gid, etc. Assim temos toda a informação para conseguirmos fazer os próximos passos da autorização de permissões.

Primeiramente, verificamos se quem está a aceder ao ficheiro é o dono do mesmo, caso seja verdade terá logo permissões, como seria de esperar. Se não for o dono, é feita uma procura pelo dono na base de dados. Se o nome for encontrado é retornado o email do dono ao qual será enviado um email com informações como: quem está a tentar aceder ao ficheiro, qual o ficheiro em questão e um código que terá que introduzir para desencadear

uma autorização. Uma vez introduzido o código, para que o *filesystem* saiba que o código foi inserido é chamado um **método GET**, que será explicado mais à frente, **de 5 em 5 segundos até atingir os 30 segundos de tempo limite**. Se o código recebido do método for igual ao gerado então será desencadeado uma autorização para aquele utilizador aceder ao ficheiro em questão, chamando a *system call open*, caso contrário é retornado erro.

Deste modo o dono consegue controlar quem pode aceder ao seu ficheiro.

1.2 Base de dados de utilizadores

Para conseguirmos manter registo de todos os utilizadores e da forma de os contactar criamos um ficheiro com uma formatação específica **"Username | endereço de email"**.

Com uma formatação *standard*, como a escolhida por nós, conseguimos facilmente procurar e retirar o endereço de email associado a um utilizador. Por acharmos que a informação guardada não é sensível, pois apenas guarda o nome de utilizador, que é público para qualquer utilizador do sistema operativo (tal como a informação em */etc/passwd*), e o endereço de email, uma forma de contactar o utilizador, não aplicamos nenhum mecanismo de encriptação.

No entanto, achamos importante restringir os acessos a esse ficheiro, ou seja, ter uma definição adequada de permissões para o mesmo. Definimos então que apenas o dono do ficheiro terá acesso privilegiado, ou seja, *read* e *write* e os restantes, tanto o *group* e *other* apenas poderão ter permissões de *read*. Utilizando o comando ***chmod*** conseguimos definir essas permissões.

- ***chmod u=rw info***, apenas atribui permissões de *read* e *write* ao *user* dono.
- ***chmod go=r info***, apenas atribui permissões de *read* ao *group* e *other* (ou, por exemplo, ***chmod go-wx info*** e ***chmod go+r info*** teria o mesmo efeito.)

Assim apenas o dono do ficheiro poderá definir quais os utilizadores registados para o *filesystem*.

1.3 Geração do Código

No que diz respeito ao código que o utilizador terá que inserir, geramos um código com 20 caracteres, contendo letras maiúsculas, minúsculas e números distribuídos de uma forma aleatória. Para conseguirmos fazer uma seleção aleatória dos caracteres a usar no código utilizamos a função ***rand*** **e** ***srand*** standards de C. No entanto, sabemos as implicações que tem a utilização destas funções, já que não geram números verdadeiramente aleatórios.

Inicialmente a ideia seria a geração criptográfica de números pseudo aleatórios. Tentamos implementações usando como recurso ***/dev/urandom***, e também a biblioteca openSSL e a sua implementação de números random, ***RAND_bytes***. No entanto, nem sempre conseguimos transformar o conjunto de bytes resultantes de ambas implementações numa string perceptível para ser usada como código na linguagem C, e quando aparentemente conseguíamos, o resultado não era sempre percetível. Então como último recurso decidimos implementar usando as funções *rand* e *srand* e aumentar o tamanho do código com o objetivo de ser mais "seguro".

1.4 Comunicação do código

O mecanismo escolhido para realizar a comunicação do código de segurança, ao dono do ficheiro a ser acedido, foi **correio eletrónico**. Escolhemos esta opção, pois foi a que nós permitiu o envio gratuito sem a necessidade de um *free trial*. Outras opções como por exemplo, envio de SMS, impõe limite de mensagens gratuitas.

Para implementarmos o serviço de email recorremos à **biblioteca *curl***, que nós permite, utilizando o protocolo SNMP, enviar mensagens para destinatários indicados. O servidor SNMP utilizado foi o do Gmail - *smtp://smtp.gmail.com:587*.

Para conseguirmos ter um ligação segura ao servidor de email, e como opção disponibilizada pelo *curl*, utilizamos a verificação de certificados SSL, indicando onde o certificado CA se encontra. Aquando da conexão ao servidor remoto, o certificado indicará que o servidor é confiável e que a conexão pode acontecer. O certificado encontra-se junto com o projeto na pasta *SSL-Certificate*. Tal como seria de esperar, é também usado o protocolo SSL/TLS para obter uma comunicação segura tanto das credenciais da conta de email a ser utilizadas, bem como do *payload* da mensagens para o remetente.

A seguir conseguimos ver o que um utilizador receberá se tentar aceder a um ficheiro do qual ele não é dono.

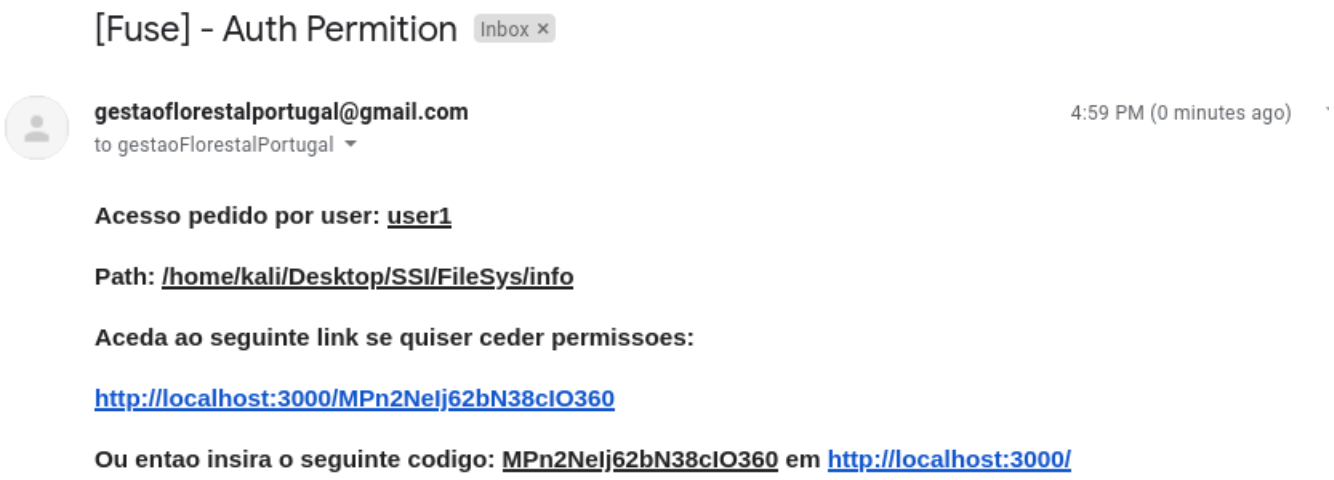

Figura 1: Email recebido por um utilizador.

1.5 Servidor com interface web

De modo ao utilizador poder introduzir o código recebido seguimos a sugestão presente no enunciado e construímos uma **interface web suportada por uma API REST**. Para tal utilizamos React e Node para construir um servidor que disponibiliza uma interface web ao utilizador, e para a API REST recorremos à *framework* de Java - Dropwizard.

Como podemos ver na imagem apresentada na secção 1.4, ao utilizador é apresentado um link que o redireciona para uma página web. Essa página web foi desenhada para apenas permitir a introdução de um código pelo utilizador. O utilizador pode então:

- Aceder a *http://localhost:3000* e introduzir o código.
- Aceder a um link do tipo *http://localhost:3000/**CODE***, onde o ***CODE*** representa o código, e automaticamente é preenchido o espaço definido para o código.

Depois do passo de introdução do código, o utilizador poderá submeter o código. O código é guardado num recurso na API. Esta disponibiliza dois métodos:

- Um de **GET**, *http://localhost:8080/permition*, que fornece o código guardado no recurso naquele momento.
- Um de **POST**, *http://localhost:8080/permition/codigo*, que ao receber um código insere o valor no recurso código.

Para evitar que possa ser utilizado *brute force* para obter o código correto e assim desbloquear o acesso ao ficheiro, foi implementado um mecanismo que apenas permite no máximo 3 inserções de códigos, originando, posteriormente, uma espera de 5 segundos. Recorremos a esta solução por acharmos que este poderia ser um foco de ataque, mas também não quisemos definir um tempo de espera demasiado grande, o que impediria que acessos concorrentes a ficheiros bloqueassem depois de 3 POSTs no recurso, deixando os utilizadores mais tempo à espera. Com o tempo de 5 segundo conseguimos limitar o número de tentativas possíveis durante os 30 segundos de espera no acesso a um ficheiro, originando aproximadamente, apenas 18 possíveis tentativas até atingir o tempo limite. No que toca à experiência do utilizador, não deverá ser impactante já que esse é o tempo do código percorrer toda a infraestrutura até ser verificado.

Este mecanismo foi implementado aquando de cada POST à API, sendo só autorizado se forem respeitadas as restrições, caso contrário é enviado como resposta 403-*Forbidden*.

Estes dois métodos disponibilizados foram criados com propósitos previamente definidos. O método GET permite ao *filesystem* realizar, continuamente, GETs à API durante 30 segundos ou até receber um código válido (correspondente ao código gerado pelo próprio). O segundo método é utilizado pelo servidor *frontend* para indicar à API uma inserção no recurso (sempre que for inserido e submetido um código).

Figura 2: Interface web apresentada ao utilizador para inserção do código.

É possível encontrar mais imagens da interface na secção 7.1

2 CVE e CWE

Durante o desenvolvimento do projeto tentamos ter em consideração as fraquezas mais comuns presentes no CWE. Algumas delas encontram-se na categoria "Error Conditions, Return Values, Status Codes" e "Handler Errors", onde tentamos evitar que não acontecessem "Unchecked Return Value", tratando sempre os valores de retorno das funções, e sempre que algum desses *returns* fosse de erro, tratá-lo de modo a não afetar o resto do programa. Tal pode ser visto no ficheiro "passthr.c" onde todos os possíveis resultados são tratados. O mesmo acontece nas funções dos outros ficheiros, dando também valores significativos no *return* e tratando possíveis exceções.

Um exemplo de uma vulnerabilidade comum que tomamos em consideração é a possibilidade de ataque de *brute force* permitir dar permissões a qualquer *user* que tenha acesso ao *filesystem*. No entanto, através do mecanismo explicado na secção anterior conseguimos mitigar esta possível vulnerabilidade.

3 Instalação

Para facilitar a instalação de alguns *packages* decidimos recorrer a ***scripts bash*** onde são executados alguns comandos de instalação referentes à API REST e ao servidor web. Para isso basta executar ***bash runInstal.sh***. De realçar, no entanto, a necessidade de ter instalado o ***node*** **e** ***npm*** para poder correr o servidor web. Já em relação ao *filesystem* apenas é necessário proceder à instalação normal da biblioteca *libfuse* (na linguagem C) tal como indicado nos passos do *github* e também ter instalada a **biblioteca** ***libcurl4-openssl-dev*** necessária para o envio do email e chamada dos requests à API REST.

4 Descrição dos componentes

Na pasta do projeto encontram-se 3 pastas e 4 *scripts bash* (Um deles já foi explicado anteriormente, e os restantes serão explicadas na secção Run). A primeira pasta **"FileSys"** contém todos os ficheiros referentes ao *filesystem*. Isto é, o ficheiro **"passthr.c"** é onde se encontram definidas todas as operações possíveis do *filesystem* e a própria main. No ficheiro **"email.c" e "email.h"** encontra-se a função referente ao envio do email. No ficheiro **"getCode.c" e "getCode.h"** encontra-se a chamada para realizar o pedido de GET à API. No ficheiro **"verify.c" e "verify.h"** encontram-se as funções para gerar o código de autorização, e a função de acesso ao ficheiro de base de dados. O ficheiro **"info"** é onde se encontram os dados dos utilizadores registados. Temos também o ficheiro **"config.h"** que tem alguns defines necessários para o projeto. Nas pasta **"rest-dropwizard"** temos o projeto referente à API REST em java. E na pasta **"WebServer"** temos o servidor web em Node e React.

5 Run

Para facilitar a testagem criamos **3 *sripts bash*** que inicializam todos os recursos, isto é, o *filesystem*, a API REST e o servidor web. Para além disso é necessário **aceder ao ficheiro config.h** localizado na pasta "FileSys" e **alterar o define PATH_TO_PROJECT para o path onde o projeto se encontra**, para que o *filesystem* consiga ter acesso ao ficheiro dos utilizadores, e o certificado SSL para envio do email. Posterior a esta alteração apenas é necessário **executar os *scripts bash*** que se encontram na pasta raiz do projeto. Começamos por executar ***bash runRest.sh*** que iniciará a API REST. De seguida, inicia-se o servidor web executando ***bash runWebServer.sh*** e depois executamos ***bash runFileSys.sh*** que procede à criação do *filesystem*. De realçar a necessidade de em ***/tmp/*** **criar a pasta "filesys1"**, onde será criado o *filesystem*. Neste último ficheiro, ao correr o executável do *filesystem* pusemos a flag -f, o que permite que seja visível qualquer *print* feito pelo *Filesystem*. Decidimos deixar esta flag pois achamos que com os prints conseguimos apresentar informação relevante, como quem está a aceder, quem é o dono do ficheiro, quando é acedida a base de dados, quando é enviado o email e quando é permitido o acesso ou são excedidos os 30 segundos.

Como seria de esperar também temos uma **makefile** que é usada nos *scripts bash*.

6 Conclusão

Em suma, este trabalho teve como principal objetivo complementar os mecanismos de controlo de acesso de um sistema de ficheiros tradicional do sistema operativo Linux com um mecanismo adicional de autorização de operações de abertura de ficheiros.

Após a realização deste projeto as principais dificuldades passaram pela implementação de certos aspetos na linguagem C. Em jeito de melhoria num possível trabalho futuro poderia-se dar mais ênfase na criação de um código de autorização mais robusto e mais seguro. Concluindo, o balanço deste projeto é positivo e pensamos ter atingido todos os objetivos propostos no enunciado.

7 Anexos

7.1 Interface Web

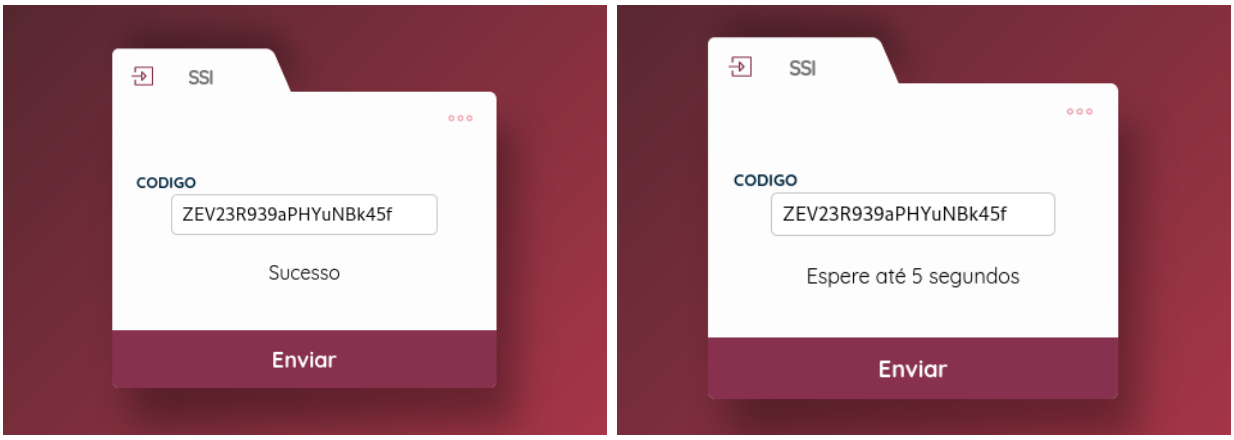

Figura 3: Interface web apresentada ao utilizador.