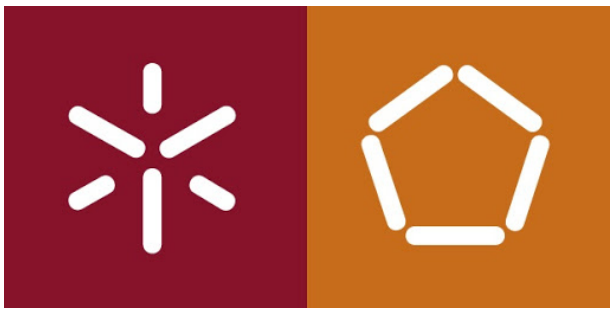

Segurança de Sistemas Informáticos
Trabalho Prático 3
Grupo 13

Daniel Regado (PG42577)
Luís Ribeiro (A85954)

Conteúdo

1 | Contextualização do Problema

A nossa resolução a este trabalho prático passou pelo desenvolvimento de 2 abordagens diferentes. Uma mais simplificada e não tão desenvolvida, onde geramos um código que é enviado para o email do *user* em específico, como sugerido no enunciado do Trabalho Prático. A outra abordagem, passa pelo uso da aplicação ***Google Authenticator***, onde é gerado um QRCode, funcionando como chave de ligação entre o telemóvel e o *FileSystem*. Mais abaixo, explicamos bem as diferenças entre os dois, apenas queremos referir que a abordagem mais trabalhada e que satisfaz melhor os objetivos pedidos é a relativa ao *Google Authenticator*. Em baixo podemos identificar as 2 abordagens referidas. A abordagem do email é identificada pela diretoria ***mailer/***, e a abordagem do *Authenticator* é indetificada pela diretoria ***authenticator/***.
Ver secção **5.1** dos **Anexos**.

1.1 Dependências e Instalação

Nesta secção segue-se as dependências de instalação para permitir o bom funcionamento dos programas. Para isto, é necessário a instalação prévia do **Python3**, tal como o seu instalador de pacotes **Python3-pip**. Dentro da instalação de pacotes do Sistema Operativo, é necessário também a instalação da interface gráfica **Python-tkinter** (*install python-tk*), para a introdução do código na abordagem do *Authenticator*, tal como a instalação das dependências do *pyfuse3*, e para isso é necessário o ***libfuse3*** e outras dependências associadas. Para isso, deve correr o comando,

```
$ sudo apt install meson cmake fuse3 libfuse3-dev libglib2.0-dev
```

As seguintes dependências referem-se aos pacotes do Python, daí a necessidade prévia da instalação do instalador de pacotes **Pip**. Para a instalação dos pacotes será necessário correr,

```
$ pip3 install <package>
```

Os pacotes necessários são,

- **pyotp**: Módulo relacionado com One Time Passwords(OTP);
- **pyfuse3**: Pacotes para a ligação com o *libfuse3*;
- **validate_email**: Permite a validação dos emails;
- **qrcode**: Geração do QRCode relativo à abordagem do *Authenticator*;
- **websockets**: Usado na implementação do *mailer*, *socket* entre o executável e a página Web de introdução do código.

1.2 Inicialização das Abordagens

Antes da explicação específica de cada abordagem, vamos primeiro referir as semelhanças das abordagens. No desenvolvimento de ambas, decidimos criar um script que irá criar um ficheiro encriptado, com dados necessários para a autenticação posterior dos utilizadores autorizados. Para isso, será necessário fornecer um ficheiro de *input*, que identifica os utilizadores do sistema, tal como o seu email de referência. No caso da abordagem do *mailer*, os emails serão utilizados para criar um ficheiro encriptado */etc/.secure_filesystem_emails*, que irá associar os respetivos emails ao *UserId*. No caso da abordagem do *authenticator*, será associado ao *UserId* uma chave simétrica que será utilizada para gerar as One Time Password (OTP) necessárias para a autenticação. Neste caso, o ficheiro encriptado gerado será */etc/.secure_filesystem_keys*. O email fornecido será usado para o envio do QR Code que associa essa chave simétrica ao utilizador, e é suposto ler esse QR Code na aplicação *Google Authenticator* ou outro software Multi-Factor Authentication (MFA) compatível.

Em baixo, apresenta-se uma imagem que representa a estruturação deste conceito dentro da abordagem do *Authenticator* (Figura 2.1). Para correr este programa, basta invocar o script *init.py*, dando como argumento o ficheiro associando users e emails.

```
$ python3 dir/init.py users.txt
```

Dir terá que ser substituído pela abordagem que o utilizador usar, isto é, *mailer* ou *authenticator*.
De notar que, aquando a execução deste comando, é pedido a introdução de um email e de uma respetiva *password* correspondente a quem enviará o código ou o QRcode para o utilizador específico. Esta abordagem é para não ficar tudo *hardcoded*, e as credenciais são as seguintes:

- Email: ssifilesystemwatcher@gmail.com
- *Password*: G5MXByuUaUWL4SV

2 | Authenticator

Tendo em conta os objetivos deste trabalho prático, decidimos investir mais nesta implementação porque achamos que seria mais prática para o utilizador sem comprometer a segurança da aplicação. Aliás, tentamos ainda fazer com que esta implementação fosse mais segura do que a implementação proposta.

Esta implementação é baseada na existência de uma chave simétrica que gera One Time Password (OTP), sendo esta alterada num período de 30 segundos. A autenticação passa por o utilizador introduzir essa OTP, e assumindo que essas OTP foram geradas pela mesma chave, a OTP será a mesma, e o utilizador será autenticado. Portanto, visto que se trata de uma chave simétrica, teremos de passar essa chave para o

utilizador, para que este consiga gerar as suas OTPs.

2.1 Inicialização

De acordo com o que foi descrito na Secção 1.2, existe um script denominado *init.py* que trata de alguns passos necessários para a autenticação dos utilizadores autorizados. Tendo em conta a existência de um ficheiro de texto que associe o email dos utilizadores autorizados ao seu username, o script *init.py* irá tratar de gerar a chave simétrica, enviar o QR Code correspondente por email, e guardar num ficheiro encriptado a associação do *UserId* à chave simétrica, isto para cada utilizador especificado no ficheiro de texto dado como input. Caso este script seja executado e o utilizador já esteja registado, irá ser criada uma nova chave secreta, e é assumido que a intenção é mesmo essa, uma nova chave. Caso seja listado um utilizador que não exista no sistema, será ignorado.

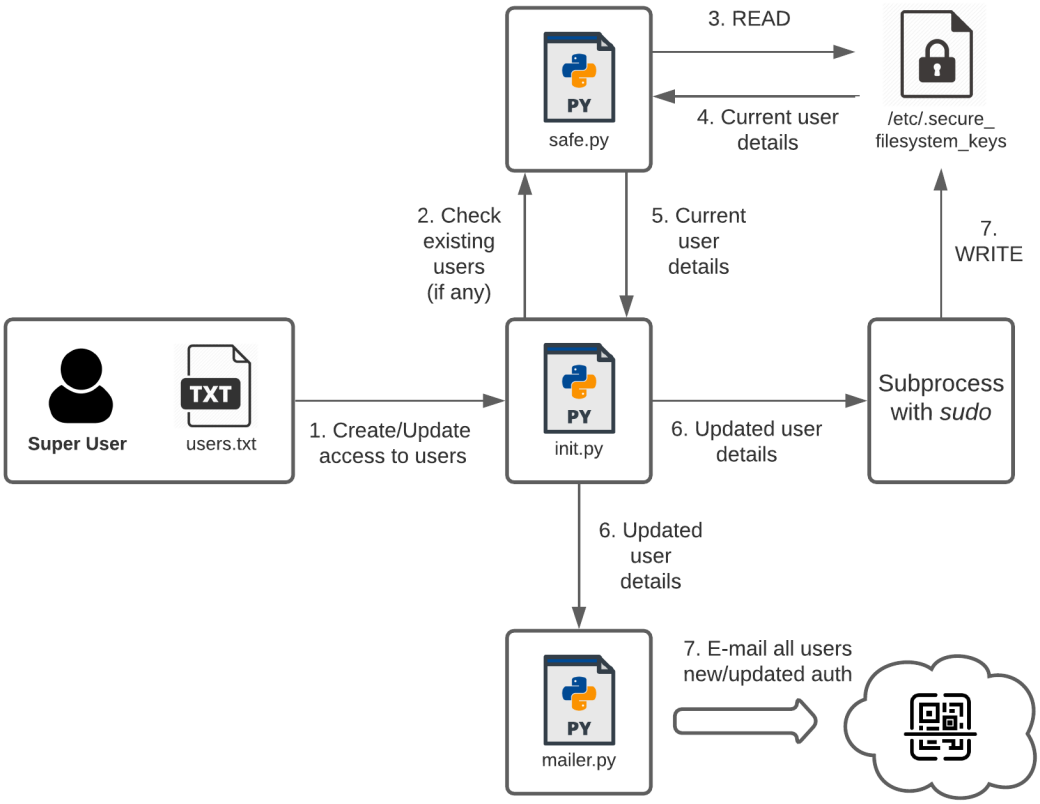

Figura 2.1: Estruturação do *init.py* no *Authenticator*

2.1.1 Geração da chave simétrica

Recorrendo ao módulo *pyotp*, geramos uma chave secreta em *base32*, que irá ser utilizada para a geração das OTPs para autenticação.

```
key = pyotp.random_base32(24)
```

2.1.2 Envio da chave simétrica

Mais uma vez, recorremos ao *pyotp* para a geração do URI, e ao módulo *qrcode* para incorporar este.

```
uri = pyotp.totp.TOTP(key).provisioning_uri(name=user, issuer_name=
                                'Secure Filesystem Authentication')
qr = qrcode.QRCode(version=1, error_correction=
           qrcode.constants.ERROR_CORRECT_L, box_size=10, border=5)
qr.add_data(uri)
```

Por fim, usamos o script *mailer.py* para enviar o email, associando a este o QR Code. Como foi referido anteriormente, é necessário numa primeira execução registar um email e palavra-passe que envie estes emails. Caso seja necessário alterar estas credenciais, basta apenas executar o script *mailer.py* diretamente.

Existem dois tipos de email, o de criação e o de renovação, sendo que o primeiro é utilizado quando um utilizador é registado pela primeira vez, enquanto que o segundo é utilizado para atualização da chave simétrica (quando reset_user = True).

```
mailer.sendQRcode(email, qrcode, username, system, reset_user)
```

2.1.3 Gravação dos dados gerados

Uma vez gerados todas as chaves secretas para os utilizadores, é necessário guardar esta informação. Para isso, recorremos ao módulo *json* para converter um dicionário numa *string*, sendo esse dicionário com *keys* *UserId* e *values* chave simétrica. De forma a encriptar essa *string*, é utilizada uma *Fernet key*, que também será necessária para desencriptação. Por último, é iniciado um subprocesso que necessita de ser executado como *sudo*, de forma a ter permissões de escrita na diretoria */etc*.

```
fernet = Fernet(fernet_key)
encrypted = fernet.encrypt(str.encode(json.dumps(users_info)))
subprocess.call(["sudo", "python3", "-c", "f=open('{}','wb'); \
       f.write({});f.close()".format(safe.DATA_PATH, encrypted)])
```

De forma a que a chave de encriptação fique segura, esta é guardada recorrendo ao módulo *keyring*, que necessita de um *backend* para guardar informação (por exemplo GNOME Keyring). Dessa forma, a *fernet key* fica acessível para posterior utilização, e apenas para o utilizador que a gerou. Além da *fernet key*, as credenciais do email utilizado para envio de emails também são guardadas recorrendo ao *keyring*.

2.2 Execução

Visto que escolhemos a implementação recorrendo ao pyfuse3, fazer *mount* de uma diretoria passa pela execução do script *passthroughfs.py*, passando como argumentos *source* e *mountpoint*:

```
$ python3 authenticator/passthroughfs.py <source> <mountpoint>
```

Analisando a seguinte imagem (Figura 2.2), conseguimos ter uma visão mais geral do fluxo aplicacional na abertura de um ficheiro.

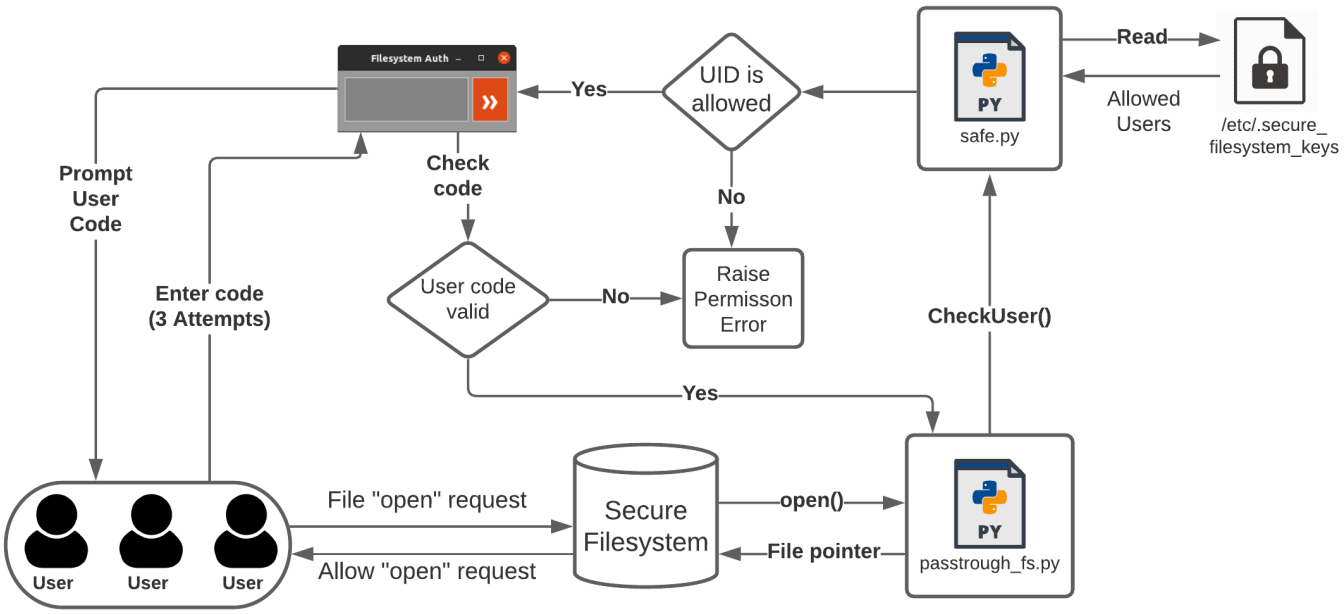

Figura 2.2: Overview do flow de execução do passthroughfs.py

No canto inferior esquerdo da figura, conseguimos ver a abstração do sistema onde está *filesystem*, representado por vários utilizadores, podendo estes estar ou não autorizados a abrir ficheiros do *filesystem*.

Como foi referido, escolhemos a implentação recorrendo ao *pyfuse3*, e usamos como ponto de partida o ficheiro *passthroughfs.py*. As alterações que fizemos foram apenas a importação do nosso módulo *safe*, e a verificação do utilizador na *function call open()*:

```
import safe
...
async def open(self, inode, flags, ctx):
    ...
    if not safe.checkUser(str(ctx.uid)):
        raise FUSEError(errno.EACCES)
    ...
```

Ou seja, durante a tentativa de abertura de um ficheiro, este trata-se de um utilizador autorizado e que introduza o código corretamente, ou então será lançado um erro de permissões. De forma semelhante à implementação em C do *libfuse3*, o objeto *ctx* é equivalente ao *get_fuse_context()*, e assim temos acesso ao *uid* do utilizador que tenta aceder ao ficheiro em causa.

2.2.1 Autorização

Na invocação da função *checkUser* no módulo *safe*, este faz leitura do ficheiro que contém informação dos utilizadores com autorização de acesso. Caso o *uid* do utilizador que tenta abrir o ficheiro não esteja incluído nos utilizadores autorizados, esta função retorna *False*, levando ao *raise* de um erro de permissões insuficientes. Caso esteja autorizado, dá-se inicio ao processo de autenticação.

2.2.2 Autenticação

Uma vez verificado que o utlizador está autorizado a abrir os ficheiros, este terá de introduzir a OTP apresentada no *Google Authenticator*. Para isso, irá aparecer uma pequena interface gráfica (Figura 2.3) para introdução do código de 6 dígitos.

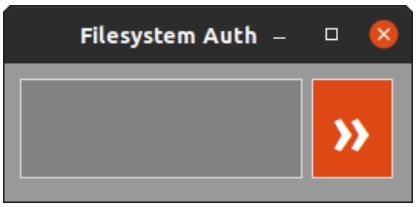

Figura 2.3: Interface gráfica para introdução de OTP

Posteriormente, esse código irá ser comparado com a OTP gerada pela chave secreta no lado do sistema. Caso coincidam, é sinal de que foram geradas pela mesma chave e que se trata mesmo do utilizador que fez a leitura do QR Code enviado por email. Essa verificação é feita com o auxilio da class *User*, no método *attempt*, essencialmente com a seguinte verificação:

```
secrets.compare_digest(self.key.now(), user_input)
```

Neste excerto do ficheiro *users.py* (linha 17), é obtida a OTP gerada pela chave guardada no sistema e depois comparada com a introduzida, recorrendo à função *compare_ digest* do módulo *secrets*. O utilizador tem 3 tentativas de introdução, caso contrário irá ser lançado o erro de associado a permissões. Caso os códigos coincidam, a função de verificação irá retornar *True*, e a execução irá continuar normalmente.
Ver secção **5.2** dos **Anexos**.

3 | Mailer

Como já referimos em cima, esta abordagem foi feita de modo a satisfazer os requisitos mínimos pedidos, daí não ser a melhor e nem a mais complexa das duas abordagens.
Primeiramente vem a etapa de **Inicialização**, onde o *init.py* é executado e é gerado o ficheiro encriptado (mencionado na secção 1.2). A fase de **Execução**, a abordagem é semelhante à de cima, no entanto devemos estar na diretoria *mailer/* para que seja possível gerar/criar a página HTML. Assim,

```
$ python3 passthroughfs.py ../<source> ../<mountpoint>
```

O mecanismo de **Autorização** é o mesmo da abordagem anterior, ou seja o *uid* do utilizador que tenta abrir o ficheiro é verificado.
Relativamente ao mecanismo de **Autenticação**, gerámos um página HTML, onde era apresentado um *form*, para que fosse possível inserir o código enviado para o *email*. Através do uso dos *WebSockets*, conseguíamos verificar o código introduzido. **Autenticação** é diferente da abordagem anterior, no sentido em que, aquando

a abertura do ficheiro, há um envio de um código aleatório de 16 caracteres para o *email* de quem tenta aceder ao ficheiro, sendo lhe apresentado a página HTML, e o utilizador tem 30 segundos e 3 tentativas para a inserção correta do código.
Ver secção **5.3** dos **Anexos**.

4 | Conclusão

Para concluir iremos agora abordar tópicos ainda não discutidos totalmente, como a Segurança, e a explicação de algumas escolhas tomadas ao longo do desenvolvimento deste trabalho prático.

Como já mencionamos, a escolha de uma diferente abordagem relativa ao pedido, foi com o intuito de aumentar a segurança destes mecanismos. Como na parte do *mailer*, tínhamos a geração e abertura de uma página HTML para que o utilizador inseri-se um código que lhe era enviado, este processo de abertura de ficheiros poderia ser explorado de forma maliciosa.

Achamos a abordagem do *Authenticator*, para além de mais segura, mais cómoda porque não é necessário ter sempre acesso à Internet para a introdução do código (ter sempre acesso ao *email*), pois o código OTP é gerado a cada 30 segundos na aplicação *Google Authenticator*.

5 | Anexos

5.1 Contextualização do Problema

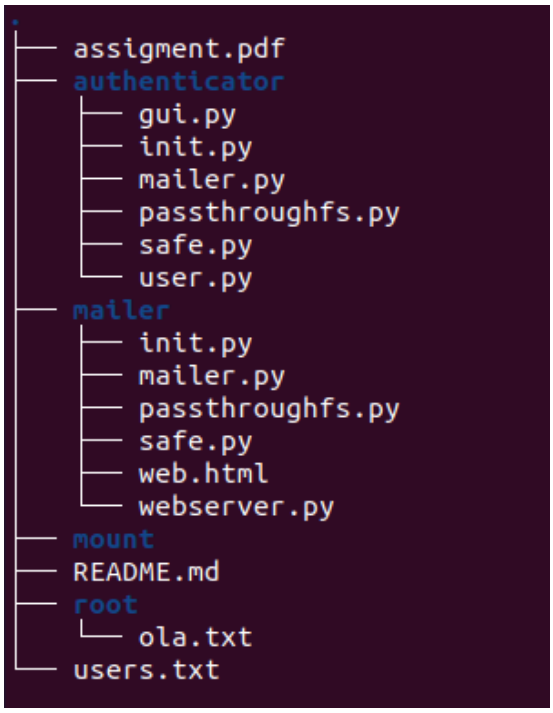

Figura 5.1: Estruturação das diretorias do Trabalho

5.2 Authenticator

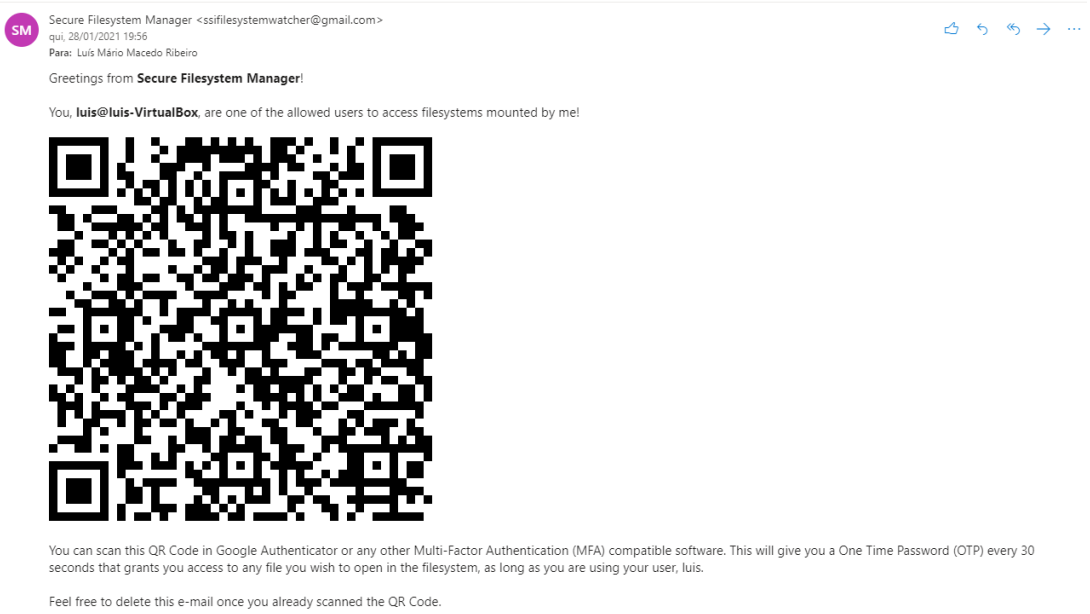

Figura 5.2: Email exemplo do envio do QRcode na versão do *Athenticator*.

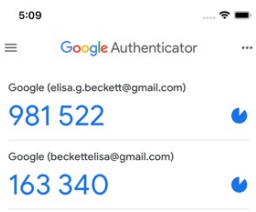

Figura 5.3: Exemplo *Google Authenticator*.

5.3 Mailer

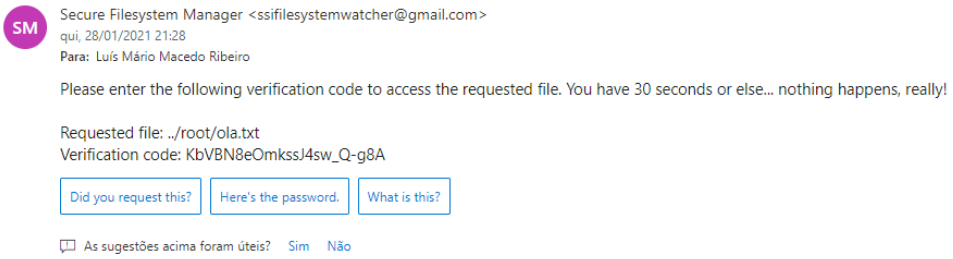

Figura 5.4: Email exemplo do envio do código gerado na versão do *Mailer*.

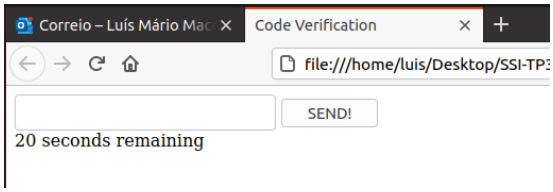

Figura 5.5: Interface gráfica para a introdução do código.