



Universidade do Minho
Escola de Engenharia

Trabalho Prático 3

Mestrado em Engenharia Informática

Tecnologias de Segurança

Nuno Daniel Simões Morais, pg46754

Tiago José Ferreira Cruz, pg47855

Henrique Daniel Freitas da Costa, pg46533

Braga, junho de 2022

Introdução

A realização deste documento tem como propósito a implementação de uma componente de software capaz de detetar modificações não autorizadas num conjunto de ficheiros considerados críticos para a segurança de um dado sistema operativo Linux, fazendo referência às abordagens e metodologias lecionadas ao longo do semestre, na UC de Tecnologias de Segurança. Esta componente deverá poder funcionar ou de forma assíncrona, como serviço, ou em tempo real, como sistema de ficheiros virtual. O nosso grupo optou, portanto, pela alternativa do sistema de ficheiros baseado na biblioteca *libfuse*, em particular, na biblioteca *pyfuse*, uma vez que a linguagem de programação utilizada foi o *python*, por ser a mais fácil de trabalhar na opinião do grupo. Quanto ao sistema operativo Linux, a opção escolhida foi o sistema virtual *Kali Linux* na qual acedemos ao mesmo através de uma máquina virtual própria.

Aspetos relevantes de segurança

Ao longo da implementação do sistema de ficheiros, fomos tendo o cuidado com os aspetos de segurança, tais como, vulnerabilidades conhecidas e fraquezas comuns das bibliotecas usadas e, consequentemente, fomos adicionando algumas medidas de segurança. Tendo em conta isso, as fraquezas tratadas foram as seguintes:

- CWE-798 - Hard Coded Credentials
- CWE-862 - Missing Authorization

Para a primeira fraqueza, desenvolvemos um script que cria o ficheiro chamado `secret.json` onde apenas o root consegue aceder, ou seja, desta forma evitamos ter credenciais no código, isto é, o segredo do Google Authenticator.

Para a segunda fraqueza, usamos a técnica da autenticação de dois fatores com o auxílio da aplicação Google Authenticator como vai ser explicado mais à frente.

Estrutura da solução

Permissoes.py

Essencialmente este script é necessário para as credenciais de autenticação do Google Authenticator, de maneira a evitar vulnerabilidades de hard coded.

Este ficheiro contém uma classe que ao ser executada, será pedido que insira as credenciais do Google Authenticator. Contudo, consideramos ainda outros ficheiros que deveriam ser mantidos seguros.

Então esta classe cria 3 ficheiros, *atributos.json*, *catfile.txt* e *secret.json* com permissões de root. Para isto utilizamos os seguintes comandos:

- `sudo chown root:root path` para o ficheiro;
- `sudo chmod 700 path` para o ficheiro.

Passthrough.py

Nesta classe utilizamos, como exemplo, o passthrough disponível no github do pyfuse, com a introdução de modificações em algumas funções e com adição de outras.

Funções alteradas:

- *getattr()*;
- *open()*.

Funções adicionadas:

- *getPaths()*;
- *loadJson()*;
- *authentication()*;
- *checkPermission()*;
- *getCriticalFileAttributes()*.

Na função *getattr()*, para além de utilizarmos os atributos de cada ficheiro, colocamos também as permissões de leitura, escrita e execução associadas a cada ficheiro e a hash de cada ficheiro e, de seguida, aplicamos todas estas informações recolhidas dentro de um dicionário para, posteriormente, transferir para um ficheiro JSON.

Na função *open()*, o objetivo é verificar permissões ao abrir ficheiros. Dessa forma, o grupo achou que seria prudente fazer a autenticação de dois fatores. Essencialmente, a função *open()* verifica se o ficheiro que está a ser aberto é crítico ou não, caso seja, é necessário verificar as permissões que o utilizador tem e quem é a pessoa que está a aceder e posteriormente pedir o código de autenticação.

A função *getPaths()* é utilizada como auxiliar para obter os *paths* dos ficheiros que precisamos para executar o programa.

A função *loadJson()* é usada para fazer *load* de um ficheiro JSON para um dicionário dict_.

Na função *authentication()* vamos gerar o código de autenticação que iremos precisar para ter acesso a ficheiros críticos.

Começamos por ir buscar o TOTP a um ficheiro de texto chamado *secret* com secret key e geramos o uri através do qual, obtivemos o código de autenticação que iremos buscar à aplicação *google authenticator*. Caso o código esteja correto, retornamos *true*; caso contrário, damos print de “Código Inválido” e retornamos *false*.

Na função *checkPermission()* verifica-se se o utilizador corrente é o mesmo que criou o ficheiro. Se sim, precisa de autenticação e retornamos *true*. Caso não seja, retornamos *false*.

A função *getCriticalFileAttributes()* chama todos os *paths* e ficheiros dentro das diretorias presentes na lista *listCriticalFolders*.

De cada diretoria passamos para cada ficheiro, onde invocamos a função *getattr()* que irá colocar dentro de um dicionário todos os atributos relativos ao ficheiro com key, o nome do ficheiro e value, os atributos.

De seguida, transferimos toda a informação para dentro de um JSON, onde posteriormente, mudamos as permissões para apenas o utilizador ter permissão de leitura, de forma a que não se possa alterar o ficheiro.

Helper.py

No script *Helper.py* temos 3 funções:

- *jsonPath()* onde definimos o path para o ficheiro JSON;
- *hashFilePath()* onde definimos o path para um ficheiro auxiliar onde é colocada a hash de cada ficheiro para depois ser possível usar na função *getattr()*;
- *getFileIds()* onde obtemos o gid e uid de cada um dos ficheiros para depois utilizarmos esta informação na função *getattr()*.

Código de acesso:

No google authenticator dá-nos algo como isto:

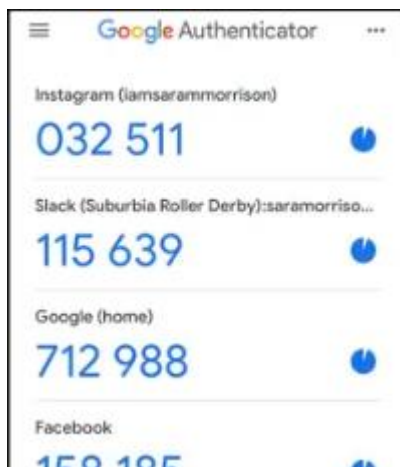


Figura 1 - Códigos de acesso do Google Authenticator

Instalação das bibliotecas

Para o funcionamento da aplicação, é necessário executar os seguintes comandos:

- pip install fusepy na root;
- sudo apt install Fuse;
- pip install pyotp.

Execução do programa

Antes de executar é necessário alterar os paths das variáveis, *secret_path*, *atr_path* e *hash_path* dentro do script *Helper.py*. Note que não são necessários paths específicos.

Depois é necessário executar o script *Permissoes.py* (necessita de execução root) tal como explicado anteriormente lhe serão pedidos as credenciais do Google Authenticator. Nós usamos as seguintes:

- Segredo: 3232323232323232
- Nome: dancrossss
- Issuer_name: dancrossss

Para executar o script *Permissoes.py* é necessário ir para a diretoria onde o ficheiro se encontra.

Para executar o ficheiro *Passthrough.py*, é necessário abrir um terminal root, pois existem ficheiros cujas permissões apenas podem ser acedidas com utilizador root, tais como, *atributos.json*, *secret.txt* e *catfile.txt*, sendo estes cruciais para o funcionamento do sistema. Após abrir o terminal, deve dirigir-se para a diretoria onde o ficheiro se encontra. Uma vez aí, o comando abaixo deve ser executado de forma a iniciar a aplicação:

python Passthrough.py / (local onde fazer mountpoint)

No nosso caso, fazemos mount para uma pasta *myMountPoint*.

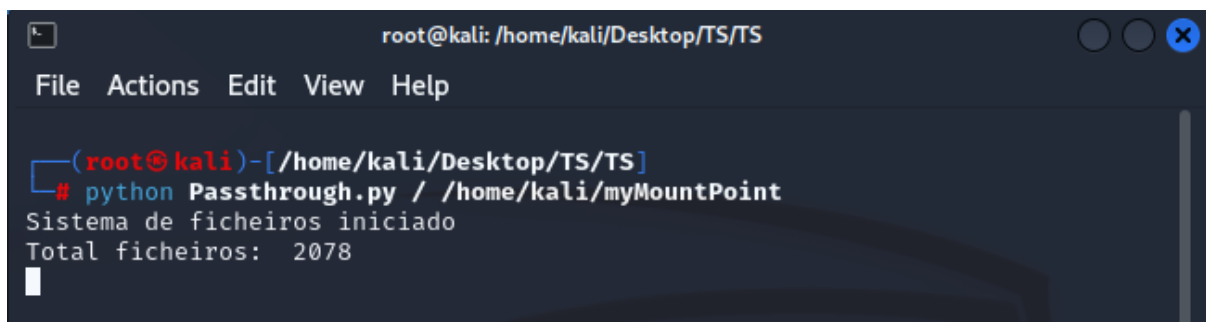
A screenshot of a terminal window titled 'root@kali: /home/kali/Desktop/TS/TS'. The window has a menu bar with 'File', 'Actions', 'Edit', 'View', and 'Help'. The terminal shows a prompt '(root@kali)-[/home/kali/Desktop/TS/TS]' followed by the command '# python Passthrough.py / /home/kali/myMountPoint'. Below the command, the output reads 'Sistema de ficheiros iniciado' and 'Total ficheiros: 2078'. A cursor is visible on the line following the output.

Figura 2 - Execução do ficheiro *Passthrough.py*

Quando o sistema é inicializado, como falado anteriormente, é executado a função *getCriticalFileAttributes()* de forma a obter os atributos e *hashs* de todos os ficheiros considerados críticos pelo grupo. Neste caso, fazemos apenas para a diretoria “*/etc*”, devido à carga de ficheiros presentes, pois, caso contrário iria demorar imenso tempo para executar o programa.

Uma vez com o sistema de ficheiros iniciado, temos de voltar a abrir uma consola (podendo ser um user normal) e entrar no sistema de ficheiros (no caso em cima, a diretoria é */home/kali/myMountPoint*). Se

a intenção for aceder a algum ficheiro considerado crítico, por exemplo (*/etc/shadow*), será feita a autenticação de dois fatores para quem quer que tente aceder ao mesmo e mediante validação ou não do código devolvemos as permissões que determinado utilizador tem.

Nas figuras abaixo é possível verificar um utilizador a tentar aceder ao ficheiro *shadow*.

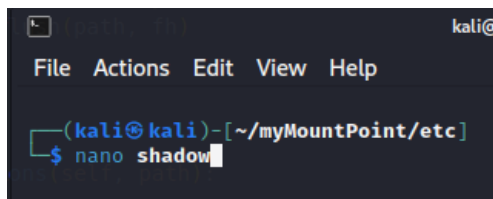


Figura 3 - Utilizador a aceder ao ficheiro shadow

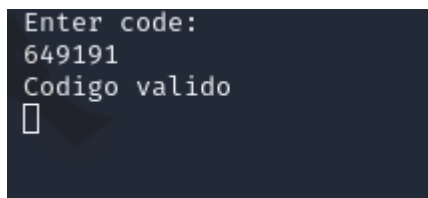


Figura 4 – Utilizador consegue aceder ao ficheiro

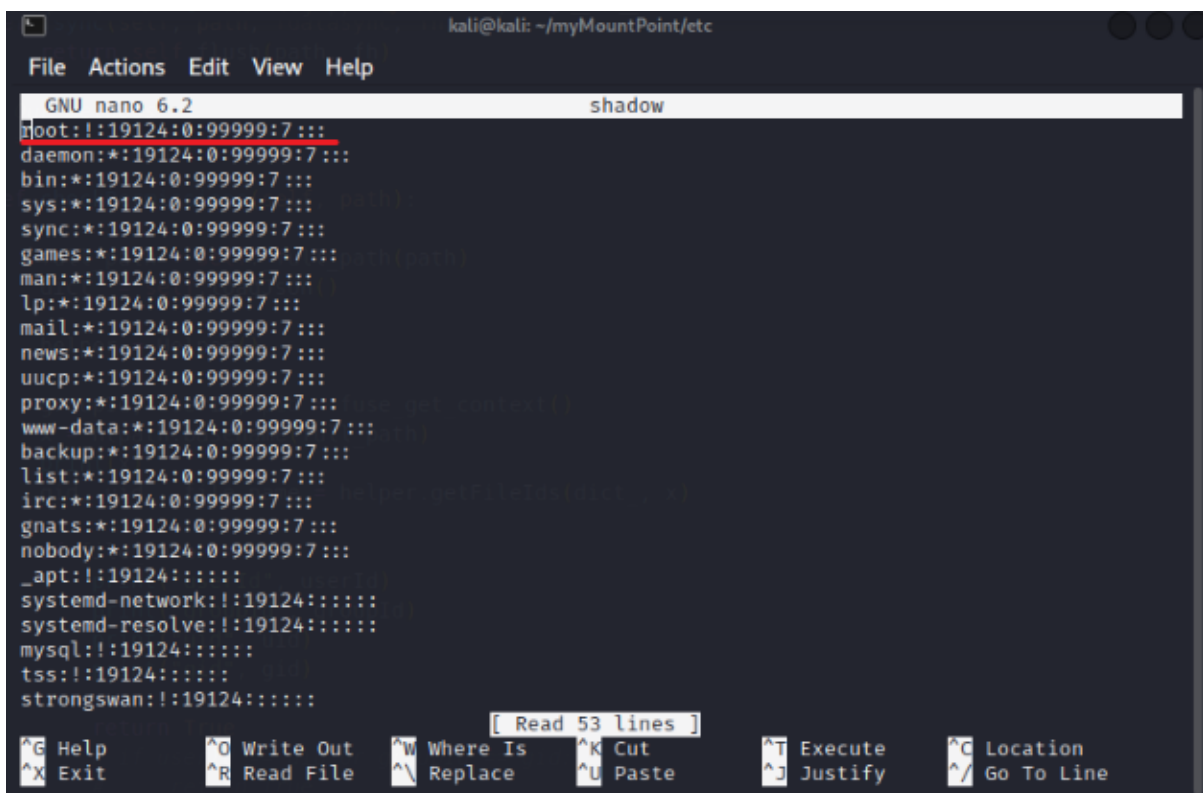
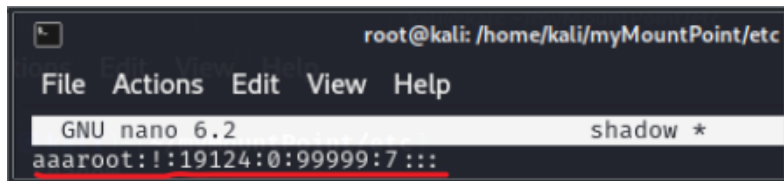


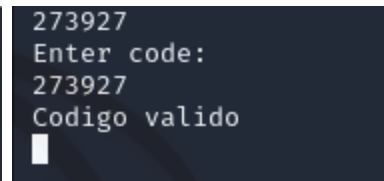
Figura 5 - Ficheiro shadow inalterado

De notar que, se o utilizador root aceder ao ficheiro, passar a autenticação e, de alguma forma, alterar o ficheiro (ver figuras 6,7 e 8), ser-lhe-á pedido novamente a autenticação de forma a validar as alterações feitas.



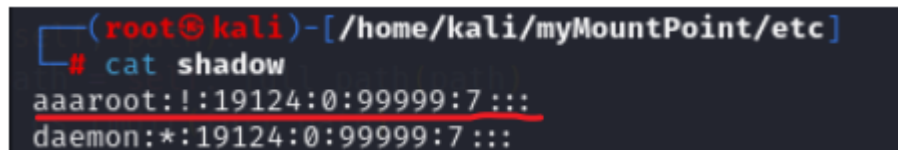
```
root@kali: /home/kali/myMountPoint/etc
File Actions Edit View Help
GNU nano 6.2 shadow *
aaaroot:!:19124:0:99999:7:::
```

Figura 6 - Ficheiro shadow alterado



```
273927
Enter code:
273927
Codigo valido
█
```

Figura 7 - Pedido de autenticação ao utilizador para finalizar alteração



```
(root@kali)-[/home/kali/myMountPoint/etc]
# cat shadow
aaaroot:!:19124:0:99999:7:::
daemon*:19124:0:99999:7:::
```

Figura 8 - Visualizar ficheiro shadow

Conclusão

Para concluir, consideramos que grande parte dos objetivos propostos foram alcançados, ou seja, foi implementado um sistema de ficheiros capaz de detetar modificações não autorizadas num conjunto de ficheiros críticos que pudessem comprometer um sistema operativo Linux. Na execução do mesmo, tivemos de ter em conta as vulnerabilidades conhecidas bem como as fraquezas mais comuns das bibliotecas utilizadas quer ao nível da autenticação, quer ao nível das credenciais. Dito isto, realçamos que o sistema desenvolvido seria capaz de cumprir todas as exigências requeridas inicialmente.