Universidade do Minho

Mestrado em Engenharia Informática

# Tecnologias de segurança

# Trabalho prático 3

# Grupo 1

Rui Carlos Azevedo Carvalho - PG47633
Daniel Barbosa Miranda - PG47123
Ana Luísa Lira Tomé Carneiro - PG46983

# Contents

# 1 Arquitetura e estrutura

## 1.1 Estrutura geral

Este projeto gera dois executáveis, sendo que, o primeiro (`monitord`) vai ser utilizado por serviços enquanto que o segundo (`mon`) pode ser utilizado por qualquer utilizador do sistema.

De facto, utiliza-se o `systemd` para correr dois tipos de serviços. O primeiro, é um serviço que é utilizado apenas uma vez e vai servir para pré-inicializar o ficheiro de registos, que mantém os ficheiros a serem vigiados assim como os metadados dos mesmos. Após este ser realizado, utiliza-se outro serviço mas que desta vez vai ser executado de forma periódica, sendo que, para isto utiliza-se um *timer* que à semelhança dos ficheiros dos serviços, tem também um ficheiro de configuração em que basicamente executa o serviço que se pretende tornar periódico indicando de quanto em quanto tempo este vai ser executado. Desta forma, através do `systemd` conseguimos realizar tanto o serviço de inicialização como o verificador da integridade dos ficheiros de forma regular.

Por conseguinte, nesta verificação de integridade caso algum ficheiro tenha sido removido ou algum dos seus dados tenha sido alterado então é escrito para um ficheiro de logs (`rsyslog`) a alteração que foi efetuada ao ficheiro.

Já o segundo executável pode ser utilizado por qualquer utilizador do sistema e serve para adicionar ou remover um ficheiro em *runtime* da lista de monitorização de ficheiros.

## 1.2 Ficheiros necessários e permissões

| Ficheiro | Descrição | Localização | Permissões |
|---|---|---|---|
| monitord-registry.txt | Ficheiro que contém informações (metadados) sobre os vários ficheiros a monitorizar. | /etc | -rw-r--r-- |
| monitord-init.txt | Ficheiro que vai ser lido inicialmente e que contém paths para ficheiros para preencher o ficheiro de registos. | /etc | -rw------- |
| monitord-logs.log | Ficheiro que armazena os logs da verificação de integridade dos ficheiros. | /var/log | -rw-r--r-- |
| monitord | Executável que vai ser utilizado pelos serviços do systemd para inicializar o ficheiro de registos ou fazer uma verificação da integridade dos ficheiros. | /opt | -rwx------ |
| mon | Executável que pode ser utilizado pelos utilizadores para adicionar/remover ficheiros da lista de monitorização. | /etc | -rwsr-xr-x |

Figure 1: Ficheiros utilizados e permissões

A tabela acima revela os ficheiros principais e necessários para o bom funcionamento de todas as componentes deste projeto, assim como a descrição de cada um deles, as localizações no *file system* e as permissões que lhe foram atribuidas.

## 1.3 Daemon

O `daemon` possui duas funcionalidades, sendo que, uma delas permite a inicialização do ficheiro de registos a partir de um ficheiro inicial (`monitord-init.txt`) que possui uma lista de *paths* de ficheiros. Cada uma das entradas do ficheiro de registos é formatada da seguinte forma:

**Path:inode:mode:num_links:size:owner:group:hash**

Obviamente para esta funcionalidade realizam-se verificações de existência dos ficheiros a partir dos seus *paths*, assim como se existe a possibilidade de escrever no ficheiro de registos.

Já a segunda funcionalidade é aquela que realiza a verificação da integridade dos ficheiros. Desta forma, é feita a verificação da possibilidade de ler o ficheiro de registos e caso o seja então é feita a leitura, linha a linha, deste mesmo ficheiro. Para cada um destas linhas extrai-se o *path* do ficheiro e utiliza-se a função `load_file` do módulo `file.c` para extrair as informações do ficheiro. Note-se que, se não for possível extrair as informações deste ficheiro, seja porque foi apagado ou o seu nome foi alterado, é enviada uma mensagem para os *logs* a constatar este facto. Caso contrário, compara-se cada uma das características do ficheiro com aquelas presentes no ficheiro de registos e caso alguma esteja diferente então escreve-se também no ficheiro de logs uma nova entrada a indicar aquilo que foi alterado.

## 1.4 Cliente

Este programa foi implementado para que seja possível a um utilizador acrescentar ou remover ficheiros que estão a ser monitorizados pelo serviço, desde de que este seja o dono do ficheiro a adicionar ou a remover.

Para se poder adicionar um ficheiro à lista de ficheiros a ser monitorizados é necessário proceder à execução do comando:

```
$ /etc/mon -a <path do ficheiro>
```

A utilização deste comando, contudo, passa por múltiplas etapas que garantem a eficiência e segurança desta operação começando por verificar se o ficheiro a adicionar realmente existe e, se existir, verifica-se se se possui permissões de leitura sobre o ficheiro de registos, cuja existência também é verificada antes de continuar. Em caso positivo, é efetuada uma consulta ao mesmo, na qual se averigua se o ficheiro já está contido nele ou não. Se estiver, significa que já está a ser monitorizado e então termina-se a operação de adição. Em caso contrário, verifica-se se o autor do comando é *root* ou *owner* do ficheiro e se sim, adiciona-se o mesmo ao ficheiro de registos.

Quanto à remoção de um ficheiro da lista de ficheiros monitorizados, o comando a utilizar é o seguinte:

```
$ /etc/mon -d <path do ficheiro>
```

O seu procedimento é bastante semelhante ao anterior em certas etapas, sendo que, mais uma vez, começa-se por verificar a existência do ficheiro a remover. Em caso afirmativo cria-se um ficheiro temporário com permissões de escrita e verifica-se se se possui permissões de leitura sobre o ficheiro de registos e em caso positivo, antes de começar a remoção propriamente dita do ficheiro garante-se que é necessário ser *root*

ou *owner* do mesmo. De seguida, percorre-se o ficheiro de registos, copiando cada linha do mesmo ao ficheiro temporário, com a exceção da correspondente ao ficheiro a remover. Após isso, abre-se o ficheiro de registos em modo de escrita e substitui-se o seu conteúdo pelo conteúdo do ficheiro temporário, terminando-se por apagar este último.

# 2 Instalação

## 2.1 Scripts

Para a execução dos serviços e programas que foram implementados foi necessário a utilização de ficheiros do tipo *service* e *timer* que vão correr no `systemctl`. Desta forma temos:

- **monitord-init.service**: Este serviço vai invocar o ficheiro `monitord-init.sh` sendo que esta vai executar o `monitord` de forma a que este leia o ficheiro `/etc/monitord-init.txt` que contém (ou nao) os ficheiros a serem rastreados logo que o serviço começa.
- **monitord.service**: Vai ser executado sempre que o `monitord.timer` o invocar e vai posteriormente executar o programa `monitord` de forma a este verificar a integridade dos ficheiros.
- **monitord.timer**: Este temporizador só vai ser inicializado após o fim da execução doi serviço monitord-init.service e vai invocar o serviço `monitord.service` de 1 em 1 minutos de forma a que esta execute o programa `monitord`.

Para utilização do serviço foi necessário a criação de diversos *scripts* que permitem a instalação, inicialização, uso e finalização do programa `monitord` e do programa do cliente `mon`. Abaixo encontram-se os *scripts* utilizados:

- **install.sh**: Este *script* é utilizado para a instalação e criação dos os ficheiros e serviços necessários para o bom funcionamento do programa. Este *script* começa por direcionar os logs criados pelo serviço para o caminho `/var/log/monitord-logs.log`. De seguida cria-se o ficheiro `/etc/monitord-registry.txt` e o `/etc/monitord-init.txt`. Além destes ficheiros é também armazenada na pasta `/etc` o programa do cliente, `mon`, sendo que, é possível que quem tente executar o programa o faça como se fosse o dono do executável. É também neste *script* que o ficheiro `monitord-init.service`, o *timer* e o `monitord.service` serão copiados para a pasta `/etc/systemd/system/` onde se encontram os restantes serviços do `systemd`.
- **start.sh**: Neste *script* dá a inicialização do serviço através da ativação e começo do `monitord-init.service` e do `monitord.timer` no `systemd`. Desta forma vamos ter o serviço `monitord-init.service` a ser executado apenas uma vez e o serviço `monitord.service` a ser executado de 1 em 1 minuto devido ao *timer* `monitord.timer`.

- **stop.sh**: Este script vai terminar e desativar todos os serviços que façam parte do systemd, sendo eles o `monitord-init.service`, `monitord.timer` e, por fim, o `monitord.service`.

- **unistall.sh** vai eliminar todos os ficheiros que foram criados para o funcionamento do programa, desde de serviços, *timers*, ficheiros, executáveis e os logs.

## 2.2 Modo de utilização

Começamos por executar a makefile para a criação dos executáveis `monitord` e `mon` através do comando:

```
$ make
```

Dentro da pasta **scripts** executar o comando de instalação para instalar o programa no sistema:

```
$ sudo bash install.sh
```

Para utilizamos o serviço podemos começar por acrescentar ao ficheiro de inicialização `/etc/monitord-init.txt` caminhos de ficheiros que pretendemos que sejam verificados pelo serviço, através do comando:

```
$ sudo vi /etc/monitord-init.txt
```

Dentro da pasta **scripts** executar o comando de inicialização dos serviços implementados:

```
$ sudo bash start.sh
```

Caso pretendemos adicionar ou remover um ficheiro do serviço utilizamos respetivamente os seguintes comandos:

```
$ /etc/mon -a <path do ficheiro>
$ /etc/mon -d <path do ficheiro>
```

Para aceder aos logs do programa ou verificar a informação sobre os ficheiros a serem vigiados utilizamos, respetivamente, os comandos:

```
$ cat /var/log/monitord-logs.log
$ cat /etc/monitord-registry.txt
```

**Nota**: O ficheiro dos logs so existirá caso o programa tenha de facto escrito pela primeira vez neste ficheiro, ou seja, o serviço verificou que um dos ficheiros que está a monitorizar foi modificado.

Caso pretendamos parar e desinstalar os serviços implementados utilizamos, respetivamente, os seguintes comandos:

```
$ sudo bash stop.sh
$ sudo bash uninstall.sh
```

# 3 Aspetos de segurança

## 3.1 Dependências e vulnerabilidades conhecidas (CVE)

A implementação deste projeto está dependente de bibliotecas externas que podem causar vulnerabilidades ao projeto. As bibliotecas e ferramentas e as respetivas versões utilizadas no projeto são o **Rsyslog 8.2112.0**, **OpenSSL 3.02** e o **Systemd 249.11**. A primeira ferramenta é utilizada como um porcessador de logs e neste caso foi utilizada para escrever os *logs* do `monitord` de forma a que utilizador consegui-se ter acesso aos mesmos mesmo com o programa a correr como serviço do `systemd`. A biblioteca `OpenSSL` foi utilizado para obter a *hash* dos ficheiros que estão a ser rastreados pelo o sistema, já a ferramenta `systemd` fornece um *array* de componentes do sistema para Linux e no caso do projeto foi utilizado para correr os servicos `monitord-init.service`, `monitord.timer` e o `monitord.service` implementados.

De seguida procedemos à verificação de vulnerabilidades conhecidas para cada uma destas dependências. De facto, para a versão utilizada do `Rsyslog` que é a mais recente, ainda não estão registadas vulnerabilidades na base de dados. Já relativamente ao `OpenSSL`, apesar deste apresentar vulnerabilidades, estas não estão relacionadas com as componentes utilizadas no desenvolvimento do *software* em questão. Por fim, a versão utilizada do `Systemd` é a mais recente, apresentando portanto, poucas vulnerabilidades conhecidas, sendo que nenhuma está relacionada com as componentes utilizadas no projeto.

## 3.2 Common Weakness Enumeration - CWE

Existem alguns tipos de fraquezas associadas ao tipo de código que foi desenvolvido neste projeto e que foram tidas em conta de modo a mitigá-las.

De facto, a fraqueza mais comum que pode ser associada a este projeto tem que ver com problemas associados à gestão de privilégios [1], sendo que, para além da definição das permissões dos ficheiros indicados anteriormente também foram feitas verificações nas operações de remoção ou adição de ficheiros à lista de monitorização, através da verificação do ID do utilizador que estava a executar o programa com o intuito de perceber se esse utilizador poderia adicionar ou remover o ficheiro em questão. Ainda nesta categoria, está descrito como fraqueza o *handling* incorreto de permissões insuficientes, sendo que, através das verificações mencionadas é possível detetar se o utilizador possui ou não permissões insuficientes e caso não tenha é lhe enviada uma mensagem de erro, ou seja, o programa não é afetado de forma negativa por este tipo de ação.

Por conseguinte, outra categoria de fraquezas tem que ver com o *handling* de ficheiros, sendo esta categoria por problemas relacionados com o tratamento dos *paths* de ficheiros ou criação de ficheiros temporários com permissões incorretas [3]. Ora, no código desenvolvido é possível verificar que o utilizador para adicionar ou remover o ficheiro introduz o seu nome/*path*. De forma a construir um *path* absoluto do ficheiro utiliza-se a função `realPath` que realiza algumas verificações sobre o nome do ficheiro introduzido, tal como se o seu path absoluto é menor que o valor da variável `PATH_MAX`, se o ficheiro de facto existe, entre outras. Assim sendo, caso o seu valor devolvido seja NULL então é possível verificar se o *path* é válido ou não. Quanto à utilização

de ficheiros temporários, é utilizada a função `tmpfile`, que cria um ficheiro com permissões de escrita e leitura para o seu *owner* cujo nome é único e é eliminado assim que se fecha o ficheiro ou o programa termina, o que leva à mitigação desta fraqueza.

Outra fraqueza insere-se na categoria de erros de auditoria ou *logging* [4] onde se indica a falta de clareza ou detalhe nos *logs* que são armazenados, assim como a introdução sensível de informação nos mesmos. Ora, nesta aplicação os *logs* que são armazenados indicam de forma clara o ficheiro que foi alterado e que componente dos seus metadados é que foi alterada, assim como a data da introdução do *log* que é automaticamente assinalada através da utilização do `rsyslog`. Desta forma, os *logs* apresentam um nível de detalhe aceitável e não revelam informações sensíveis.

Por conseguinte, outra categoria de fraquezas comuns está relacionada com *pointers* [2], mais especificamente, a utilização de *pointers* nulos ou a sua utilização antes de serem inicializados. Ora, no código desenvolvido é feita a verificação dos valores destas variáveis antes da sua utilização, como por exemplo, antes de utilizar um variável do tipo `FILE *`, sendo que, a libertação destes é sempre efetuada de modo a impedir *memory leaks*, algo que foi confirmado com a utilização da ferramenta `valgrind`.

Para além disto, existem ainda fraquezas relacionadas com a utilização de funções de APIs [5], mais especificamente na utilização de funções perigosas ou obsoletas. Desta forma, foi feita a verificação das funções que poderiam causar algum perigo, como por exemplo, a função que calcula o *hash* do ficheiro com a biblioteca `OpenSSL`, onde garantimos que a função utilizava uma função de *hash* com um nível de segurança aceitável (`SHA-256`) e que a função em si não estava *deprecated* na versão mais atual da biblioteca.

Por fim, a última categoria de fraquezas está relacionada com problemas de concorrência [6], que pode acontecer no acesso a alguns dos ficheiros mencionados anteriormente, nomeadamente se dois ou mais utilizadores tentarem eliminar ou adicionar um ficheiro ao mesmo tempo, ou o processo de verificação de integridade dos ficheiros começar quando um utilizador adiciona ou remove o ficheiro. Assim sendo, para contornar este problema utilizamos a *flag* `O_EXCL` no momento de abertura dos ficheiros com a *system call* `open` de modo a abrir os ficheiros de forma exclusiva aquela processo.

# 4 Conclusão

Na nossa opinião o projeto desenvolvido apresenta uma implementação correta do problema em questão, sendo que, é possível destacar vários pontos positivos, tais como a utilização de várias componentes da biblioteca `systemd` para criar o serviço integrado no sistema, o tipo de permissões que foram atribuídas aos vários ficheiros necessários, e a possibilidade de adicionar ou remover ficheiros em *runtime*. Para além disto, foi feita uma análise das principais fraquezas que são comuns de existirem neste tipo de projetos, sendo que, foram aplicadas diversas medidas para as mitigar, com especial destaque para os problemas relacionados com o *handling* de ficheiros e os problemas de concorrência.

Por outro lado, existem alguns pontos onde o projeto pode melhorar, tais como, a possibilidade de permitir que um utilizador possa adicionar/remover vários ficheiros com a apenas um comando ou até incluir todos os ficheiros dentro de uma diretoria utilizando uma *flag* recursiva "-r", por exemplo.

# References

[1] "CWE CATEGORY: Privilege Issues" [online]. Disponível em: https://cwe.mitre.org/data/definitions/265.html [Acedido em junho de 2022].

[2] "CWE CATEGORY: Pointer Issues" [online]. Disponível em: https://cwe.mitre.org/data/definitions/465.html [Acedido em junho de 2022].

[3] "CWE CATEGORY: File Handling Issues" [online]. Disponível em: https://cwe.mitre.org/data/definitions/1219.html [Acedido em junho de 2022].

[4] "CWE CATEGORY: Audit / Logging Errors" [online]. Disponível em: https://cwe.mitre.org/data/definitions/1210.html [Acedido em junho de 2022].

[5] "CWE CATEGORY: API / Function Errors" [online]. Disponível em: https://cwe.mitre.org/data/definitions/1228.html [Acedido em junho de 2022].

[6] "CWE CATEGORY: Concurrency Issues" [online]. Disponível em: https://cwe.mitre.org/data/definitions/557.html [Acedido em junho de 2022].