Altres FIMs

What is a FIM:

Today, most IT systems that store and process information use file-based architectures. The core operating system and applications binaries, system and application configuration data, organizational data, and logs are stored in files. These files ultimately determine how the operating system, its subsystems and hosted applications should operate; track (in log files) the actions and activities that take place across the operating system and applications and store business data. Attackers may attempt to overtake the operating system or application, steal or modify business-critical information, or manipulate log files to hide any malicious activities. This is where File Integrity Monitoring (FIM) helps, by ensuring that you're notified when such suspicious activities take place on critical files. FIM technologies typically work with one of the following approaches but regardless of approach, the end result is the same: to identify and alert you to any changes (creation, modification or deletion) to a monitored file or directory.

*** Baseline comparison,** wherein one or more file attributes will be captured or calculated and stored as a baseline that can be compared against at some future time. This can be as simple as the time and date of the file, however, since this data can be easily spoofed, a more trustworthy approach is typically used. This may include periodically assessing the cryptographic checksum for a monitored file, (e.g. using the MD5 or SHA-2 hashing algorithm) and then comparing the result to the previously calculated checksum.

* **Real-time change notification**, which is typically implemented within or as an extension to the kernel of the operating system that will flag when a file is accessed or modified. This way is what we have seen used in Falco or Audit.

Typically, a FIM is part of a complete HIDS (as we have seen previously with the Falco or Audit systems) but it can also be used as a stand-alone tool. As an example of this, here will study Inotify-tools () but other very interesting open-source FIM systems (sometimes integrated inside a complete HIDS) are:

OSSEC (https://github.com/ossec/ossec-hids)
Wazuh (https://github.com/wazuh). Its an OSSEC's fork which provides integration with ELK
Tripwire (https://github.com/Tripwire/tripwire-open-source)
Watchman (https://facebook.github.io/watchman) Més orientat a desenvolupadors CI/CD
Osquery (https://github.com/facebook/osquery). No és un FIM però aquesta suite incorpora aquesta funcionalitat dins del conjunt de possibilitats que ofereix
Stealth (https://gitlab.com/fbb-git/stealth)
AIDE (http://aide.sourceforge.net)
Samhain (https://www.la-samhna.de/samhain)
Systraq (http://mdcc.cx/systraq)

Inotify-tools:

Inotify es un subsistema del kernel Linux que provee una API con un mecanismo para monitorear eventos en el sistema de archivos. Puede ser utilizado para monitorear archivos individuales o directorios. Cuando se utiliza para monitorear un directorio, inotify retorna eventos asociados al directorio en sí mismo y a todos los archivos que contiene.

NOTA: The inotify subsystem has thrree important tunings:
"/proc/sys/fs/inotify/max_user_instances" file content impacts how many different root dirs you can watch
"/proc/sys/fs/inotify/max_user_watches" file content impacts how many dirs you can watch across all watched roots
"/proc/sys/fs/inotify/max_queued_events" file content impacts how likely it is that your system will experience a notification overflow (if it is too small the kernel will drop events)

La utilidad *inotifywait* (incluida en el paquete "inotify-tools", https://github.com/rvoicilas/inotify-tools) permite esperar cambios en archivos o carpetas haciendo uso de la API de inotify, así: *inotifywait /ruta/carpeta* o *inotifywait /ruta/fichero*

Esta herramienta puede finalizar al primer evento que ocurra (comportamiento por defecto) o monitorear de manera continua (con el parámetro *-m*); en este último caso, se puede indicar un timeout con *-t nºsegundos* para esperar ese tiempo antes de finalizar. En cualquier caso, *inotifywait* imprimirá los eventos por la salida estándar a medida que ocurran, comportamiento muy útil para usarse desde scripts. Si se añade el parámetro *-r*, en el caso de monitorizar una carpeta se hará de forma recursiva.

Es posible personalizar el formato de la salida de *inotifywait* mediante la opción *--format* ; por ejemplo, si solo interesa saber el nombre del archivo modificado/creado se puede utilizar el indicador *%f*

También es posible filtrar los eventos que se quieren mostrar mediante el parámetro *-e nombreEvento,otro,otro...* Los eventos posibles son:

access	A watched file or a file within a watched directory was read from.
modify	A watched file or a file within a watched directory was written to.
attrib	The metadata of a watched file or a file within a watched directory was modified. This includes timestamps, file permissions, extended attributes etc.
close_write	A watched file or a file within a watched directory was closed, after being opened in writeable mode. This does not necessarily imply the file was written to.
close_nowrite	A watched file or a file within a watched directory was closed, after being opened in read-only mode.
close	A watched file or a file within a watched directory was closed, regardless of how it was opened. Note that this is actually implemented simply by listening for both close_write and close_nowrite, hence all close events received will be output as one of these, not CLOSE.
open	A watched file or a file within a watched directory was opened.
moved_to	A file or directory was moved into a watched directory. This event occurs even if the file is simply moved from and to the same directory.
moved_from	A file or directory was moved from a watched directory. This event occurs even if the file is simply moved from and to the same directory.
move	A file or directory was moved from or to a watched directory. Note that this is actually implemented simply by listening for both moved_to and moved_from, hence all close events received will be output as one or both of these, not MOVE.
move_self	A watched file or directory was moved. After this event, the file or directory is no longer being watched.
create	A file or directory was created within a watched directory.
delete	A file or directory within a watched directory was deleted.
delete_self	A watched file or directory was deleted. After this event the file or directory is no longer being watched. Note that this event can occur even if it is not explicitly being listened for.
unmount	The filesystem on which a watched file or directory resides was unmounted. After this event the file or directory is no longer being watched. Note that this event can occur even if it is not explicitly being listened to.

EXERCICIS:

1.-a) Instal.la el paquet "inotify-tools" en una màquina virtual qualsevol que tinguis a mà i executa aquesta comanda: *inotifywait -m -r -e modify,attrib,close_write,move,create,delete /tmp* ¿Què estàs fent? Executa seguidament en un altre terminal la comanda *echo "Hola" > /tmp/hola.txt* vàries vegades. ¿Què veus a la sortida de *inotifywait*?

b) Digues per a què serveix el següent script abans de provar-lo. Prova'l i confirma que hagis encertat:

```
#!/bin/bash
if [[ ! -d $1 ]]
then
        echo "Has d'escriure la ruta d'una carpeta com a primer paràmetre"
        exit
fi
while [[ true ]]
do
        inotifywait -e modify,attrib,move,create,delete $1 && tar -czf $HOME/backup-$(date +"%D-%T").tgz $1
done
```

NOTA: Substituïnt la comanda *tar* per una altra com *rsync* o similar podríem implementar un Dropbox cassolà

bII) Per què a l'script anterior es fa servir un bucle infinit en comptes d'aprofitar el paràmetre *-r* de *inotifywait* ?

NOTA: En realitat es podria fer servir el paràmetre -r sense cap problema. Només caldria reescriure l'script així:

```
#!/bin/bash
if [[ ! -d $1 ]] ; then
        echo "Has d'escriure la ruta d'una carpeta com a primer paràmetre" && exit
fi
inotifywait -r -e modify,attrib,move,create,delete $1 | while read ruta accio fitxer; do
        tar -czf $HOME/backup-$(date +"%D-%T").tgz $1
done
```

c) El paquet "inotify-tools" incorpora una altra comanda anomenada *inotifywatch* que permet obtenir estadístiques. Per provar-la, primer executa en un terminal *inotifywatch -e modify,attrib,close_write,move,create,delete/tmp* ; seguidament executa en un altre terminal la comanda *echo "Hola" > /tmp/hola.txt* vàries vegades i finalment interromp l'execució d'*inotifywatch* amb CTRL+C. ¿Què veus a pantalla llavors?

Units Path:

En realitat no cal instal.lar cap paquet extra (com ara l'"inotify-tools") per poder fer ús del subsistema Inotify perquè Systemd ja incorpora una funcionalitat semblant "de sèrie" (encara que amb un grau de detall menor a l'hora de detectar events que el que oferia *inotifywait*, això sí).

Per aconseguir que Systemd detecti events en el sistema de fitxers només cal crear dos fitxers:

*Un fitxer amb un nom qualsevol però amb extensió ".path" i ubicat dins de la carpeta "/etc/systemd/system" que tingui un contingut semblant a aquest...:

```
[Unit]
Description=Monitoriza canvis en l'arxiu /etc/passwd
[Path]
PathModified=/etc/passwd
Unit=pepito.service
```

*...i un altre amb el nom que s'hagi indicat a la línia "Unit" de l'arxiu anterior (i ubicat també dins de la carpeta "/etc/systemd/system") que tingui un contingut semblant a aquest...:

```
[Unit]
Description=Executa una determinada comanda
[Service]
Type=oneshot
ExecStart=/usr/bin/play beep.wav
```

A la secció [Path] del primer fitxer es defineix la ruta del fitxer o carpeta que es vol que Systemd -també a través d'Inotify- monitoritzi per detectar-hi canvis. Depenent del tipus de canvi que es vol detectar, es poden utilitzar diferents directives, les quals es llisten a continuació:

PathExists=/ruta/fitxer/o/carpeta	This checks whether the path in question is created. If it does, the associated service unit is activated.
PathExistsGlob=/ruta/fitxer/o/carpeta	This is the same as the above, but supports file glob expressions ("comodines")
PathChanged=/ruta/fitxer/o/carpeta	This watches the path location for changes inside. The associated service unit is activated if a change is detected, but only when the watched file is closed.
PathModified=/ruta/fitxer/o/carpeta	This watches for changes like the above directive, but even before watched file is closed (so it's more granular)
DirectoryNotEmpty=yes	This directive allows systemd to activate the associated service unit when the directory is no longer empty.
Unit=nomUnit.service (o target)	This specifies the service unit to activate when the path conditions specified above are met. If this is omitted, systemd will look for a .service file that shares the same base unit name as this unit.
MakeDirectory=yes	This determines if systemd will create the directory structure of the path in question prior to watching.
DirectoryMode=0xxx:	If the above is enabled, this will set the permission mode of any created path components. By default is 755.

A més, cada fitxer "*.path" ha d'estar associat (a través de la seva línia "Unit", com ja s'ha dit) a un fitxer "*.service" (normalment homònim) que s'activarà automàticament quan s'hagi detectat algun canvi. Aquesta activació ve en forma d'execució de la comanda indicada a la seva línia "ExecStart".

NOTA: Hi ha diferents maneres d'executar la comanda indicada a "ExecStart" però la que farem servir nosaltres sempre serà la de tipus "oneshot", ja que és la que reprodueix més fidelment el comportament de l'execució d'una comanda estàndar en un terminal

Un cop creat els dos arxius anteriors, per posar en marxa la monitorització caldrà executar *systemctl start nomUnit.path* Si, a més, al final de l'arxiu "*.path" hi aparegués una secció *[Install]* incloent la línia *WantedBy=graphical.target* (per sistemes gràfics) o *WantedBy=multi-user.target* (per sistemes només de text), la monitorització llavors es podria activar automàticament mitjançant *systemctl enable nomUnit.path* En qualsevol cas, sempre es pot veure l'estat de la monitorització amb *systemctl status nomUnit.path*

NOTA: La unit "service" no ha d'estar en marxa perquè ja es posarà en marxa automàticament quan la unit "path" detecti l'event monitoritzat!

EXERCICIS:

1.-a) ¿Què fan aquests fitxers? Prova'ls (has de tenir instal.lat el paquet "sox" per poder executar *play*)

/etc/systemd/system/ojolagarto.path

```
[Unit]
Description=Misteri
[Path]
Unit=ojolagarto.service
PathModified=/etc/passwd
```

/etc/systemd/system/ojolagarto.service

```
[Unit]
Description=Misteri 2
[Service]
Type=oneshot
ExecStart=/usr/bin/play beep.wav
```

b) ¿Què fan aquests fitxers? Prova'ls

/etc/systemd/system/ojolagarto2.path

```
[Unit]
Description=Misteri 3
[Path]
Unit=ojolagarto2.service
PathExistsGlob=/home/usuari/Baixades/*.iso
MakeDirectory=true
```

/etc/systemd/system/ojolagarto2.service

```
[Unit]
Description=Misteri 4
[Service]
Type=oneshot
ExecStart=/usr/bin/systemd-cat echo "Ha aparegut una iso a la carpeta 'Baixades' de l'usuari 'usuari'"
```

c) Què hauries de fer per a què la monitorització de qualsevol dels dos exemples anteriors fos permanent a cada reinici?