

SSII - PAI 1

Implementar HIDS

Andrea Carosi
Marcel Espejo Cuenca

Contenido

Introducción:	2
Planteamiento inicial:	2
Desarrollo de cada subproblema:	3
Creación de las máquinas virtuales:.....	3
Recolectar las rutas de cada archivo:.....	3
Conexión entre máquinas:	4
Estructura de datos:	4
Función de verificación:	5
Generación de una respuesta por parte del servidor:	6
Validación en el cliente:	7
Debilidades típicas:	7
Colisión:.....	7
Almacenamiento:	7
Alarmas tardías:	7
Resultados y tests:	8
Modo de empleo:	9
Conclusiones y comentarios:	10

Introducción:

A continuación, se presenta un resumen del desarrollo llevado a cabo para implementar un HIDS desde cero.

La primera decisión importante del proyecto ha sido decantarnos por una de las dos opciones que se nos proponen en el enunciado:

- HIDS ya creado:
Una opción es utilizar una herramienta ya implementada y enfocar el proyecto a aprender a configurar un HIDS.
- Crear nuestro propio HIDS:
Otra opción es crear desde cero por lo que el proyecto consistirá en implementar toda la funcionalidad posible de un HIDS.

Finalmente hemos optado por crear nuestro propio HIDS por los siguientes motivos:

Aunque utilizar un HIDS ya existente puede ser muy útil por motivos prácticos, hemos decidido enfocarnos más en la opción que nos permite manejar cómo funciona por dentro el sistema y así comprenderlo lo mejor posible más allá de como configurar algo que ya se ha hecho por otra persona o una empresa.

Planteamiento inicial:

Seguir la estructura planteada en clase siguiendo los pasos que subdividen nuestro problema en tareas menores:

- 1) Creación de dos máquinas virtuales que simulan una situación en la que un cliente (una de las máquinas) que solicitará a un servidor (la otra máquina) la revisión de la integridad del sistema de archivos original.
- 2) Crear un programa capaz de recorrer el sistema de archivos desde un directorio determinado y generar un archivo “.csv” con la ruta de cada archivo y el Hash asociado a cada archivo.
- 3) Crear una conexión capaz de transmitir archivos “.csv” entre ambas máquinas y crear los elementos necesarios para la comunicación, en algún momento debe poderse establecer dicha comunicación de forma automática una vez al día.
- 4) Decidir y tener implementada una estructura de datos adecuada para almacenar en la máquina servidor los datos provenientes del cliente.

- 5) Implementar una función de verificación en la máquina servidor capaz de comprobar si alguno de los Hash que llegan del cliente es distinto de los originales que están almacenados.
- 6) Generar un nuevo archivo “.csv” en el servidor que almacene por cada Hash que le ha enviado el cliente, dicho Hash, un MAC (mediante el Hash del Hash anterior y un token generado aleatoriamente y que ambas máquinas conocen) e información relativa a si el Hash transmitido es correcto.
- 7) En el cliente se debe implementar de nuevo una función de verificación usando el MAC que obtiene el cliente con sistema de archivos y el MAC que llega del servidor.

Desarrollo de cada subproblema:

Creación de las máquinas virtuales:

Se crean dos máquinas virtuales para realizar la simulación usando la herramienta VirtualBox.

Recolectar las rutas de cada archivo:

Para tratar este problema escribiendo el programa en Python nos ayudamos de las librerías “hashlib” y “csv”, tras recorrer el sistema de archivos de forma recursiva obtenemos el fichero “.csv” con el formato mostrado en la figura 1.

```
tuple2.csv
1 /Users/M/Desktop/US/Cuarto/SSII/PAI1/pruebas/SSII/ciccio,e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b785
2
3 /Users/M/Desktop/US/Cuarto/SSII/PAI1/pruebas/SSII/project.py,4710cf4275fd5acbb119ddb94922bd09ba8a0704cfc8dfc6090a92
4
5 /Users/M/Desktop/US/Cuarto/SSII/PAI1/pruebas/SSII/tuple2.csv,e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991
6
7 /Users/M/Desktop/US/Cuarto/SSII/PAI1/pruebas/SSII/server/conf.py,e43e9a4cdd398249203b08e85e474fa66531db72a2bc459be06
8
9 /Users/M/Desktop/US/Cuarto/SSII/PAI1/pruebas/SSII/server/deserialize.py,dfebb116d5937ad14b5c6d509dae207260228574f5a7
10
11 /Users/M/Desktop/US/Cuarto/SSII/PAI1/pruebas/SSII/server/prueba,16ce118d48a4e0120e2306a63f609372844794aee4a74d032f4
12
13 /Users/M/Desktop/US/Cuarto/SSII/PAI1/pruebas/SSII/server/rbt.py,69cfa53f8a64366dc6115a876c72a21bba25c2fb78970a4214e3
14
15 /Users/M/Desktop/US/Cuarto/SSII/PAI1/pruebas/SSII/server/response.csv,829b285eb6d06af94cc42801b2ee138b9fb70db29773
16
```

Conexión entre máquinas:

Para empezar, abrimos un socket en el cliente y el servidor (IP+puerto), la primera vez el servidor abre un socket y se queda escuchando las peticiones de conexión del cliente, para ello usamos la librería socket de Python. Una vez establecida la conexión se envía un archivo “.csv” que por cada línea se almacene la ruta y el Hash del archivo. Una vez terminada la transferencia del archivo se cierra la conexión para evitar cualquier tipo de ataque en ese puerto.

Tras procesar la información en el servidor, se crea otra conexión, pero esta vez es el cliente quien se queda escuchando y el servidor es quien envía datos necesarios para realizar la verificación en el cliente, pero el proceso es el mismo que se ha llevado a cabo para la conexión anterior.

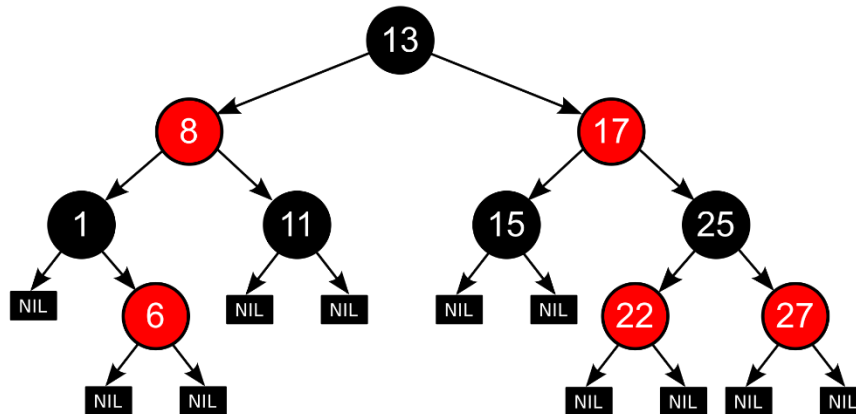
Mediante las librerías Schedule (pip install schedule) y time de Python se consigue que las conexiones de cliente a servidor y de servidor a cliente se establezcan automáticamente cada día (86400 segundos). Para realizar tests el tiempo se ha reducido a 10 s. Durante algunas pruebas nos hemos percatado de que a veces falla la conexión ya que el servidor tarda unos milisegundos en realizar el procesamiento de los datos y cuando este trata de enviar datos al cliente después ya se ha cerrado la conexión por parte del cliente, por tanto, el tiempo que tarda el servidor en realizar la segunda parte de la conexión debe ser de al menos un segundo inferior a la del cliente. Otra solución podría ser dejar siempre abierto el puerto del servidor y dejar que el cliente se autentique en cada conexión, pero por falta de tiempo y ya que no es necesario para este proyecto, hemos decidido dejarlo implementado como se ha descrito.

Estructura de datos:

Para almacenar la información y tener acceso rápido y eficiente a la misma hemos tenido en cuenta la sugerencia del enunciado, pero la complejidad en el peor de los casos a la hora de realizar una búsqueda en un árbol binario normal es de $O(n)$, por ello hemos buscado otras estructuras alternativas en forma de árbol. La mejor opción que hemos encontrado es un Van Emde Boas Tree ya que tanto para operaciones de búsqueda como inserción la complejidad es $O(\log \log n)$, pero comprender el funcionamiento de este árbol e implementarlo costaría más tiempo del que tenemos disponible para realizar el proyecto. La solución por la que nos hemos decidido finalmente ha sido el Red-Black Tree, el cual mediante el almacenamiento de un bit extra de información por cada nodo del árbol que indica su “color” es capaz de rebalancear las ramas cuando se inserta un elemento evitando así el problema que supone el árbol binario simple y pudiendo hacer consultas con una complejidad de $O(\log n)$.

La forma de reequilibrar el árbol es mediante normas que conciernen a los “colores” de los nodos, las normas son las siguientes:

- Cada nodo es negro o rojo.
- Todos los nodos hoja (NIL) son negros.
- Un nodo rojo no tiene un hijo rojo.
- Dado cualquier nodo, todo camino desde ese nodo hasta las hojas tiene el mismo número de nodos negros.



Otro problema que nos encontramos a la hora de tener la información guardada en el árbol es que no queremos tener el árbol cargado en RAM todo el tiempo, si hubiese un corte de alimentación en el equipo perderíamos todos los datos y nos sería complicado si no imposible recuperar los datos originales. La solución es serializar el árbol mediante la librería “pickle” de Python.

Función de verificación:

Una vez hemos obtenido el fichero “.csv” del cliente con todas las rutas y sendos Hash necesitamos tener de algún modo un programa que compare los Hash almacenados en nuestro árbol con los que se nos han enviado. Para ello la solución que hemos planteado ha sido crear una función dentro de la propia implementación del Red-Black Tree, usando como parámetro de entrada una línea del csv se encuentra el nodo que contiene la ruta indicada y en ese mismo momento se compara el Hash enviado por el cliente con el que se encuentra almacenado junto a la ruta en el árbol.

Una vez hecha la función nos planteamos la posibilidad de que desde el cliente se añadieran nuevos archivos, por ello implementamos en la función la posibilidad de insertar nuevos archivos con su Hash en caso de no encontrarlo en el árbol.

La función debe retornarnos una tupla de valores dos booleanos y una cadena, donde el primer valor indicará en caso de valer “True” que el Hash consultado es correcto y “False” en otro caso, para el segundo valor, “True” nos indica que la consulta actual ha sido añadida al árbol ya que es nuevo desde la consulta anterior, por último, se retorna el Hash almacenado en el árbol, esta información nos es útil para el siguiente paso.

Esta función ha sido nombrada en el código fuente como “validation_and_search”.

Generación de una respuesta por parte del servidor:

Para responder al cliente, el servidor debe ser capaz de generar un nuevo “.csv” que almacene los MAC, para ello hemos decidido que esta acción debe llevarse a cabo al mismo tiempo que se realiza la lectura del “.csv” y se consulta cada línea con la función de validación que anteriormente hemos descrito.

Por cada línea del “.csv” del cliente, se debe realizar la consulta usando la función de validación, de la cual obtenemos los dos valores booleanos de los que hablamos en el apartado anterior, los cuales serán la información extra que vaya incluida en el “.csv” de respuesta más adelante, tras la comprobación de cada línea se realiza un Hash al Hash retornado por la función de validación junto con el token generando así el MAC, con esto tenemos todos los datos necesarios para crear la nueva línea del “.csv” de respuesta cuyo formato es:

Hash del archivo enviado por el cliente, MAC, its_ok, its_new

Donde “its_ok” (valor booleano) indica si los Hashes coinciden a la hora de comprobarlo en el servidor e “its_new” (valor booleano) indica si ha sido añadido nuevo en el árbol.

```
response.csv
1  "e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
2  ",f4a93e5c7bec6e9e79be2097f4b1a727428b112939975d20f1d357e6c194979f,True,False
3
4  "4710cf4275fd5acbb119ddb94922bd09ba8a0704cfc8dfc6090a9232a6f2743
5  ",8c8bf267babebdef32795489fc9fe669e15de2106dc620de9ab67af42c9faf,True,False
6
7  "e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
8  ",f4a93e5c7bec6e9e79be2097f4b1a727428b112939975d20f1d357e6c194979f,True,False
9
10 "e43e9a4cdd398249203b08e85e474fa66531db72a2bc459be06332580184d7ba
11 ",4ac1c716772f4374ef5e0a98cd405224574f6458e37ef374ab4ab1511b05b1d3,False,False
12
13 "dfebb116d5937ad14b5c6d509dae207260228574f5a7834d63ae009d8b5c3bbd
14 ",27bc059f009c15dab009550913667533edb6d6fcd1a09d4f5b78e92bec3d7bfb,True,False
15
16 "16ce118d48a4e0120e2306a63f609372844794eaae4a74d032f49bfaafa7baa4
17 ",b77b860f3668c7bbbc131e6205f623e2ff4b9b40fb791744e3f714d52fb20daa,False,False
```

Validación en el cliente:

Una vez obtenido el MAC que nos proporciona el servidor usamos la misma función de hashing en el cliente con el Hash que ha obtenido el cliente y el mismo token para obtener otro MAC y se realiza la comparación entre ambos, en caso de ser distintos se advierte de la modificación del archivo, en caso de que el servidor nos advirtiera explícitamente que el archivo ha sido modificado pero los MAC coincidiesen también se comunica.

Debilidades típicas:

Colisión:

En nuestro HIDS es extremadamente complicado que encontremos una colisión ya que la función de Hash que utilizamos es SHA-256 teniendo 2^{256} posibles combinaciones.

Almacenamiento:

Evitamos almacenar los Hash originales en ningún lugar de la máquina del cliente, de ese modo si el sistema de archivos del mismo se ve comprometido en esa máquina no habrá posibilidad de que desde ese extremo se modifiquen nuestros Hash originales. Tratando de solucionar este problema nos encontramos con que nuestras comunicaciones entre cliente y servidor podrían llegar a ser interceptadas por un atacante, por ello cliente y servidor usaran un token para obtener un MAC, este token jamás viajara por la red ya que es introducida directamente en la máquina cliente y en el servidor a mano.

Alarmas tardías:

Este es un problema que no podemos abordar si queremos ceñirnos al enunciado ya que el proceso se lleva a cabo solo una vez al día, podríamos solventarlo aumentando la cantidad de veces que se realiza el proceso, pero podría suponer un estorbo para el funcionamiento normal del cliente. Otra solución podría ser la de implementar el HIDS de forma que cada vez que se requiere algún archivo del cliente se compruebe la integridad del archivo, pero de nuevo podría ralentizar el cliente.

Resultados y tests:

A continuación, se muestran algunas imágenes para mostrar algunas capturas realizadas durante el proceso de desarrollo en una sola máquina, cliente y servidor son dos directorios distintos y la comunicación se produce usando la ip 127.0.0.1 y puertos que estén libres.

Durante algunas etapas del desarrollo se han hecho pruebas usando el propio directorio de trabajo del proyecto como ruta a examinar por el HIDS pero esto causa errores debido a que los archivos de esta ruta mutan durante el proceso cambiando su contenido y por tanto su tamaño, el Hash producido, ...

```
kali@kali:~/SSII$ sudo python3 client.py
[+] Connecting to 127.0.0.1:22222
[+] Connected.
Sending tuple.csv: 1.88kB [00:00, 322kB/s]
[*] Opening the connection for receiving the response...
[*] Listening as 127.0.0.1:22221
[+] ('127.0.0.1', 37372) is connected.
Traceback (most recent call last):
  File "/home/kali/SSII/client.py", line 117, in <module>
    execution()
  File "/home/kali/SSII/client.py", line 105, in execution
    rev_connection(CLIENT_HOST,CLIENT_PORT,BUFFER_SIZE,SEPARATOR)
  File "/home/kali/SSII/client.py", line 63, in rev_connection
    filesize = int(filesize)
ValueError: invalid literal for int() with base 10: '2556e3b0c44298fc1c149afb4c8996fb92427ae41e464
9b934ca495991b7852b855,04203fdd2411d65bc9ade30184dafb6df40806ea799e924131468e7554c9e85f,True,False\
r\nd42625da6b802152154eb47f6a23b0da2c99a910b71abedcb3e'
```

Más adelante se habla de la necesidad de usar el super usuario para usar el HIDS, pero aquí se muestra una captura de pantalla que nos adelanta el problema de un error de permisos en un directorio protegido.

```
kali@kali:~/SSII$ python3 client.py
Traceback (most recent call last):
  File "/home/kali/SSII/client.py", line 117, in <module>
    execution()
  File "/home/kali/SSII/client.py", line 103, in execution
    file_definition()
  File "/home/kali/SSII/client.py", line 81, in file_definition
    walk_on_paths(path1)
  File "/home/kali/SSII/client.py", line 46, in walk_on_paths
    writing(file_paths)
  File "/home/kali/SSII/client.py", line 36, in writing
    writer.writerow([data] + [hash_file(data)])
  File "/home/kali/SSII/client.py", line 19, in hash_file
    with open(filename,'rb') as file:
PermissionError: [Errno 13] Permission denied: '/etc/shadow-'
```

Por último, se aprecia en una de las ejecuciones de prueba con un “print” que muestra la generación del archivo que viaja del cliente al servidor conteniendo la ruta y el Hash asociado a cada ruta.

```

kali@kali:~/SSII/server$ python3 conf.py
[*] Listening as 127.0.0.1:22222
[*] ('127.0.0.1', 43354) is connected.
Receiving tuple.csv: 1.88kB [00:00, 216kB/s]
[*] The first connection has been established
[*] Initializing data ...
/home/kali/SSII/tuple.csv,e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855

/home/kali/SSII/MAC.py,d42625da6b802152154eb47f6a23b0da2c99a910b71abedcb3e932709c95442a

/home/kali/SSII/response.csv,fcc13fa16112535bed934601429f9781038208e18b9c8ce597b2030710d43a1d

/home/kali/SSII/client.py,1602612dfdf801bf8521942c591f2b689cec71d0322a4525c15fde892a4292c0

/home/kali/SSII/__pycache__/MAC.cpython-39.pyc,f71f2765c118d2a0e59cd503f6bb747c310fc4bfee5be29cbe3cb35eb9769969

/home/kali/SSII/__pycache__/project.cpython-38.pyc,3c9ac5f3f725459f8afdd99f02748157f69165a8be69e3f224aa73b64f1820d0

/home/kali/SSII/__pycache__/MAC.cpython-38.pyc,f8d68963a7faf32f0082e0c9dcb8b80392eb7fd8a6046111c4385f8e25945bb8

/home/kali/SSII/server/conf.py,fc10175a9a4ed25dc1c10654cb6c80d545849419134cd904cd73e858c875e

/home/kali/SSII/server/tuple.csv,eb77cfa030c2f9568f4e4e16a28a419fc24fb27e5cc1787c34231f4a5e184b8b

/home/kali/SSII/server/response.csv,62d118c8363abebc2c1f4160f5d3d532593a22b178990178855768335d3f266a

/home/kali/SSII/server/rbt.py,13e8d2d6bd88ab4e8606cd58dc5154f50acac61fec04f07b225909e091d1689f

/home/kali/SSII/server/RBTbyteStream,60b25a23e39ecbb695529c1f271f88a6e9c0daecf2100f1cdf848cccd26c9ff

/home/kali/SSII/server/deserialize.py,dfebb116d5937ad14b5c6d509dae207260228574f5a7834d63ae009d8b5c3bbd

/home/kali/SSII/server/__pycache__/rbt.cpython-38.pyc,3d20e81b548585c1edd32c5eca7510eff4b9c78513caabc809bdd1d5717829b1

/home/kali/SSII/server/__pycache__/rbt.cpython-39.pyc,3a1dbc3c2506264f54adb2c38b2fdb14bd52b1b111387f86eaf965a19ba2ffaf

```

Nuestro HIDS ha sido probado también usando una máquina virtual de Kali para simular el cliente y una máquina virtual Ubuntu (debido a que es rápido de instalar) para simular el servidor.

Modo de empleo:

El proceso empieza ejecutando el archivo Python conf.py del servidor el cual hará que el servidor se quede esperando al cliente, es entonces cuando en el cliente ejecutamos client.py y el proceso está automatizado a partir de aquí (aconsejamos ejecutar ambos archivos como administrador o superusuario).

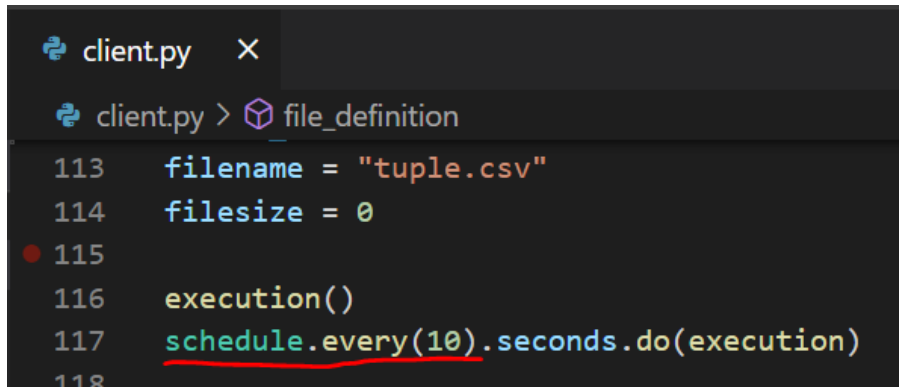
Debido a que no ha dado tiempo a hacer más la forma de cambiar la ruta que queremos que sea analizada por el servidor es cambiando el código directamente, en client.py se debe cambiar la variable path1 por la ruta requerida.

```

client.py X
client.py > file_definition
65 progress = tqdm.tqdm(range(filesize), f'Receiving {filename}')
66
67 with open(filename, "wb") as f:
68     while True:
69         bytes_read = client_socket.recv(BUFFER_SIZE)
70         if not bytes_read:
71             break
72         f.write(bytes_read)
73         progress.update(len(bytes_read))
74     client_socket.close()
75     s.close()
76
77 def file_definition():
78     csvclear()
79     path1="/etc/"
80     walk_on_paths(path1)
81     filesize = os.path.getsize(filename)
82

```

También puede cambiarse desde el código el tiempo en segundos que tarda en hacerse la comprobación (por defecto está en 10 segundos)



```
client.py x
client.py > file_definition
113 filename = "tuple.csv"
114 filesize = 0
115
116 execution()
117 schedule.every(10).seconds.do(execution)
118
```

Este proceso debería realizarse mediante un archivo de configuración, pero por falta de tiempo no se ha realizado.

Conclusiones y comentarios:

Debido a que el proyecto está dedicado a la confidencialidad de la información y que no hay suficiente tiempo, no se han implementado otras medidas de seguridad que serían altamente recomendables como la ocultación del token tanto en cliente como en servidor, codificar los logs y el borrado de los archivos auxiliares producidos tras las comunicaciones entre cliente y servidor...

Es importante hacer que los scripts de nuestro HIDS deban ser ejecutados, leídos y modificados solo por super usuario para evitar que alguien no propietario de las máquinas manipule el protocolo descrito.