SECURITY TEAM 11

Javier Hidalgo García
Juan Pedro Hurtado Masero
Nicolás Sibello Litrán

SEGURIDAD EN SISTEMAS INFORMÁTICOS E INTERNET

PAI-1 HIDS

SISTEMA DE DETECCIÓN DE INTRUSOS (HIDS) PARA ALMACENAMIENTO MASIVO BASADO EN VERIFICADORES DE INTEGRIDAD

Universidad de Sevilla
Escuela Técnica Superior de Ingeniería Informática

CONTENIDO

INTRODUCCIÓN

Problema planteado

Un enfoque clásico de la seguridad de un sistema informático siempre define como principal defensa del mismo sus controles de acceso (desde una política implantada en un cortafuegos hasta unas listas de control de acceso en un router o en el propio sistema de ficheros de una máquina), en muchos casos, esos controles no pueden protegernos ante un ataque. Desde un pirata informático externo a nuestra organización a un usuario autorizado que intenta obtener privilegios que no le corresponden en un sistema, nuestro entorno de trabajo no va a estar nunca a salvo de **intrusiones**.

Llamaremos intrusión a un conjunto de acciones que intentan comprometer la integridad, confidencialidad o disponibilidad de un recurso; analizando esta definición, podemos darnos cuenta de que una intrusión no tiene por qué consistir en un acceso no autorizado a una máquina: también puede ser una negación de servicio. A los sistemas utilizados para detectar las intrusiones o los intentos de intrusión se les denomina **sistemas de detección de intrusiones** (Intrusion Detection Systems, IDS), cualquier mecanismo de seguridad con este propósito puede ser considerado un IDS, pero generalmente sólo se aplica esta denominación a los sistemas automáticos (software o hardware): es decir, lo habitual (y lógico) es que a la hora de hablar de IDSes no se contemplen el resto de los casos.

Cualquier sistema de detección de intrusos ha de cumplir algunas propiedades para poder desarrollar su trabajo correctamente: el IDS ha de ejecutarse continuamente sin nadie que esté obligado a supervisarlo. Otra propiedad, y también como una característica a tener siempre en cuenta, es la *aceptabilidad* o grado de aceptación del IDS; al igual que sucedía con cualquier modelo de autenticación, los mecanismos de detección de intrusos han de ser aceptables para las personas que trabajan habitualmente en el entorno. Por ejemplo, no ha de introducir una sobrecarga considerable en el sistema (si un IDS ralentiza demasiado una máquina) ni generar una cantidad elevada de falsos positivos (detección de intrusiones que realmente no lo son) o de logs, ya que entonces llegará un momento en que nadie se preocupe de comprobar las alertas emitidas por el detector.
Una tercera característica por evaluar a la hora de hablar de sistemas de detección de intrusos es la *adaptabilidad* de este a cambios en el entorno de trabajo. Como todos sabemos, ningún sistema informático puede considerarse estático: todo cambia con una

periodicidad más o menos elevada. Si nuestros mecanismos de detección de intrusos no son capaces de adaptarse rápidamente a esos cambios, están condenados al fracaso.

Un sistema de detección de intrusos basado en máquina (host-based IDS o HIDS) es un mecanismo que permite detectar ataques o intrusiones contra la máquina sobre la que se ejecuta. El IDS basado en host o **HIDS** monitorea las características de un host y los eventos que ocurren en él en busca de actividades maliciosas o sospechosas. Un host es un equipo o dispositivo conectado a la red. El HIDS puede identificar tanto el tráfico malicioso que entra en el host como el que origina en el propio host y que un sistema de detección basado en red no podría detectar. En definitiva, un HIDS engloba:

- Detectar y detener los ataques directos potenciales, pero no realizar un análisis en busca de malware.
- Identificar ataques potenciales y enviar alertas, pero no detener el tráfico.
- Combinar las funcionalidades de aplicaciones antimalware con protección de firewall.
- Es un sistema sin agente que analiza los archivos en un host en busca de posible malware.

Políticas y controles de seguridad

La política a seguir por la empresa es la siguiente:

"Debe verificarse diariamente la integridad de los ficheros binarios, de imágenes y directorios de los sistemas informáticos críticos y las aplicaciones de la organización y dar cuenta mensualmente al ISG de la organización de los resultados diarios de la verificación"

Solución propuesta

Script Phyton: HIDS.py

El archivo HIDS.py es el script "main" del proyecto. El resto de los archivos aportan funcionalidades que son importadas a este script. Por tanto, cuando se quiera ejecutar el proyecto para hacer prue
bas, este es el script que hay que ejecutar. La ruta protegida por defecto es una carpeta llamada "Protected" en el directorio de usuario, pero esta ruta se puede modificar fácilmente junto con muchos otros parámetros que hemos incluido en un archivo de configuración del cual daremos detalles más adelante.
La estructura que hemos escogido para almacenar las rutas de los archivos y sus respectivos hashes es un árbol binario de búsqueda:

```
22  class BinaryTree:
23      def __init__(self, ID):
24          self.ID = ID
25          self.path = None
26          self.hash = None
27          self.left = None
28          self.right = None
29
```

De esta manera ahorramos muchísimo tiempo para la búsqueda de un archivo que ha sido atacado. El script *hids.py* contiene varias funciones:

- **getFileHash** (path, buffer_size): esta función obtiene el hash de un archivo, utilizando el tamaño de un *buffer* que por defecto es de 64Kb. Se obtiene el hash mediante el algoritmo *sha256*.

```
def getFileHash(path, buffer_size = 65536):
    sha256 = hashlib.sha256()
    try:
        with open(path, 'rb') as f:
            while True:
                data = f.read(buffer_size)
                if not data:
                    break
                sha256.update(data)
        return sha256.hexdigest()
    except OSError as e:
        print("No se ha podido abrir el siguiente archivo:\n{}".format(path))
        raise e
```

- **getAllFilesInDirectory** (mainPath, files): devuelve una lista con todas las rutas de los archivos a proteger.

```
def getAllFilesInDirectory(mainPath, files = []):
    for myPath in os.listdir(mainPath):
        newPath = mainPath+'/'+myPath
        if not (os.path.isfile(newPath)):
            getAllFilesInDirectory(newPath)
        else:
            files.append(newPath)
    return files
```

- **createBST**(ids, files, a, b): función que crea el árbol binario balanceado (es decir, el hijo izquierdo y el derecho de cada nodo son arboles del mismo tamaño), a partir de una lista ordenada con todos los *ids* y rutas de ubicación de los ficheros.

```
def createBST(ids, files, a, b):
    if a > b: return    #Caso base de la recursividad
    mid = (a+b)//2      #Calcula el punto medio de la lista de ids
    root = BinaryTree(mid)
    root.left = createBST(ids, files, a, mid-1)
    root.right = createBST(ids, files, mid+1, b)
    root.path = files[mid]
    root.hash = getFileHash(files[mid])
    return root
```

- **searchFileById** (root,Id): función que realiza la búsqueda binaria en el árbol generado.

```
def searchFileById(root, ID):
    if ID == root.ID:
        return root
    elif ID < root.ID:
        return searchFileById(root.left, ID)
    elif ID > root.ID:
        return searchFileById(root.right, ID)
```

- **checkIntegrity** (tree,ids): esta función comprueba si ha sido comprometida la integridad de los archivos que se le pasan. Para un fichero, primero realiza la búsqueda binaria en el árbol y calcula el *hash (sha256)*. Después compara ambos *hash* para, en caso de haber cambiado en el periodo estipulado, anotar en el *log* que la seguridad del archivo ha sido comprometida.

```
def checkIntegrity(tree, ids):
    for i in ids:
        node = searchFileById(tree, i)
        newHash = getFileHash(node.path)
        if newHash != node.hash:
            nameFile = os.path.basename(node.path)
            logging.debug(nameFile)    #si y solo si
```

La configuración del .log se realiza con el método *logging.basicConfig()* del módulo *log*.

```
#Configuración de la gestión del Log
logging.basicConfig(filename='.\\registro.log', format='%(asctime)s %(message)s', level=logging.DEBUG)
```

Script Enviar_email.py

Poseemos también otro script de *python* llamado **envia_email.py**, este script nos permite enviar un correo a los empleados de la empresa informándoles del reporte de ataques a la integridad mensualmente en el directorio especificado mediante una función llamada *envia()*. Esta función tiene como objetivo el envío de un mensaje de correo electrónico formado y que contiene el fichero adjunto *registro.log* con la información que, mensualmente, habrá que enviar al ISG de la organización procedente. Para ello, primero

se comprueba que el registro existe, después se forman los atributos y el cuerpo del mensaje; y el propio registro como fichero adjunto. Tras esto, se realiza el envío del mensaje desde el servidor SMTP, con el usuario indicado a los destinatarios propuestos en las configuraciones previas, ubicadas en *PARAMETERS.conf*. Hemos utilizado la librería *smtplib* y diferentes características de *MIME* para dar formato al mensaje de correo electrónico.

Params.py

Este *.py* se encarga de traducir el archivo **PARAMETERS.conf** de manera que sea legible para todas las funciones que necesiten de estos parámetros, de esta manera nos queda un programa que tiene una parametrización muy sencilla y simple.

- **loadHids**() : traduce los diferentes parámetros de *PARAMETERS.conf* que se necesitan para *hids.py*

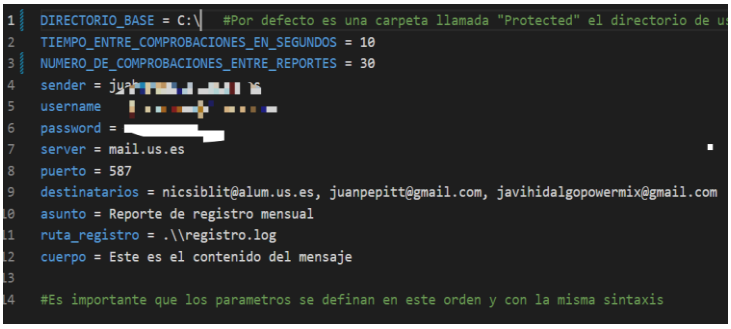
```
def loadHIDS():
    try:
        with open("PARAMETERS.conf", 'rt') as f:
            #obtenemos los 3 primeros parámetros del archivo de configuración
            #dividiendo la línea por el caracter '=', quedándonos con el segundo trozo,
            #quitando los comentarios de la linea (dividiendo por el caracter '#'),
            #y quitando los espacios con la función strip()
            params = [l.split('=')[1].split('#')[0].strip() for l in f.readlines()[0:3]]

        if params[0] == "default":
            DIRECTORIO_BASE = str(Path.home()) + "/Protected"
        else:
            DIRECTORIO_BASE = params[0]

        return (DIRECTORIO_BASE, float(params[1]), int(params[2]))

    except OSError as e:
        print("No se ha podido abrir el fichero de configuración")
        raise e
```

- **loadMail**() : traduce los diferentes parámetros de *PARAMETERS.conf* que se necesitan para *envia_email.py*

```
def loadMail():

    try:
        with open("PARAMETERS.conf", 'rt') as f:
            params = [l.split('=')[1].split('#')[0].strip() for l in f.readlines()[3:12]]
            destinatarios = [d.strip() for d in params[5].split(',')]

        return (params[0], params[1], params[2], params[3], int(params[4]), destinatarios, par

    except OSError as e:
        print("No se ha podido abrir el fichero de configuración")
        raise e
```

PARAMETERS.conf

Archivo que guarda todos los parámetros que necesita nuestro programa:



- Directorio Base: directorio a proteger de ataques. Puede ser una ruta, o se puede escribir *default,* indicando que queremos que se proteja una carpeta llamada Protected, ubicada en nuestro directorio de usuario.
- Tiempo en segundos: Tiempo entre cada comprobación de integridad. Esta cantidad de tiempo equivaldría a un día en el escenario propuesto.
- Número de comprobaciones entre reportes: Especifica cada cuántos días ficticios se envía por correo un reporte que contiene el archivo *.log*.
- Sender: emisor del correo.
- Username: usuario del emisor.
- Password: contraseña del emisor.
- Server: servidor SMTP.
- Puerto: puerto SMTP de envío.
- Destinatarios: cuentas de correo de receptores.
- Asunto: asunto del correo.
- Cuerpo: cuerpo del mensaje de correo electrónico.

PRUEBAS/TEST

Los parámetros escalados a las pruebas que hemos realizado para ver que todo funciona correctamente son los siguientes:

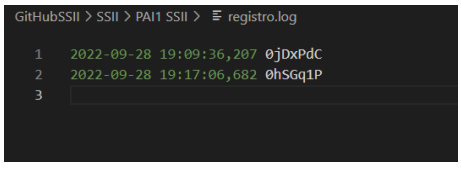
- 10 segundos corresponde a 1 día y 5 minutos (30*10) corresponde a 1 mes.

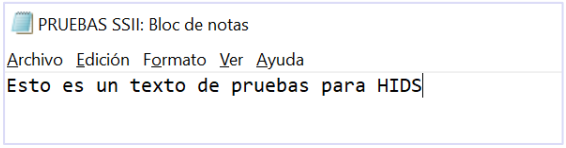
```
DIRECTORIO_BASE = C:\Users\equipo\Downloads\prueba\prueba  #Por defecto es una carpeta llamada "Protected" el directorio de usuario
TIEMPO_ENTRE_COMPROBACIONES_EN_SEGUNDOS = 10
NUMERO_DE_COMPROBACIONES_ENTRE_REPORTES = 30
sender = ju[redacted]
username [redacted]
password = [redacted]
server = mail.us.es
puerto = 587
destinatarios = nicsiblit@alum.us.es, juanpepitt@gmail.com, javihidalgopowermix@gmail.com    #Separados por comas
asunto = Reporte de registro mensual
ruta_registro = .\\registro.log
cuerpo = Este es el contenido del mensaje
```

Añadiendo los parámetros anteriores, al arrancar el programa se crea un árbol binario de 1000 archivos que, en este caso, el tiempo de creación es de ~35 segundos.

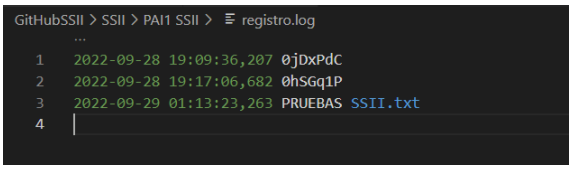
```
[Running] python -u "c:\Users\nicos\Desktop\PAI1\SSII\SSII\PAI1 SSII\Script\hids.py"
Construyendo arbol binario de busqueda de 2149 archivos
Hecho en 6.6166905 segundos!
A partir de este momento los archivos de la ruta C:\Users\nicos/Protected estan siendo protegidos
```

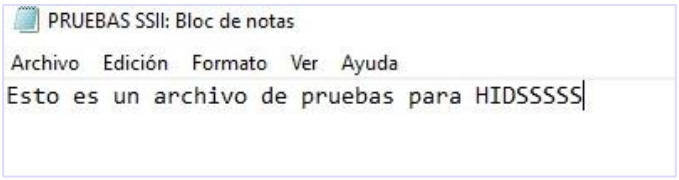
El siguiente paso será modificar un archivo de ese directorio para comprobar si se están detectando ataques a la integridad y se registra en el archivo .log:





Antes de la modificación del archivo X

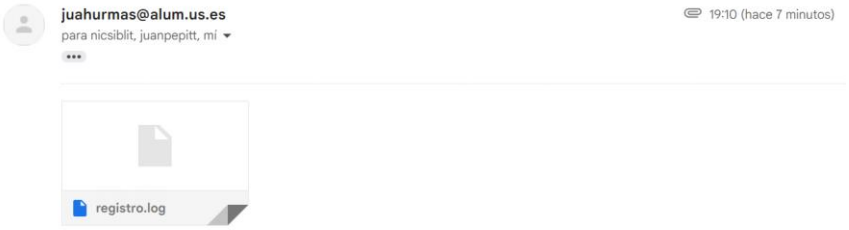




Después de la modificación del archivo X

Si planteamos el escenario en el que, con el programa en ejecución, se vulnera la integridad de un fichero (en nuestro ejemplo, "*PRUEBAS SSII.txt*"), la hora y fecha del evento se anotará en el log. Una vez verificada la vulneración por el usuario, se procede a restaurar el archivo vulnerado y no se volvería a anotar en los días posteriores. En el caso de no restaurar el fichero modificado, cada día se seguirá anotando que el fichero "*PRUEBAS SSII.txt*" ha sido vulnerado (junto con los nuevos archivos atacados, si los hubiera).

Ahora, revisando el buzón de correo que hemos definido como uno de los destinatarios, podemos observar cómo se ha recibido el fichero *registro.log*, donde se encuentra la entrada con el nombre el archivo modificado (más las anteriores, si las hubiera).



Además de las pruebas anteriormente mencionadas, hemos realizado la ejecución del proyecto en un entorno Linux con los mismos resultados. Por otra parte, hemos configurado el sistema HIDS *Tripwire* en ese entorno por si el cliente tuviera preferencia en cuanto a sistema operativo.

Repositorio del grupo: [github/SecTeam11]

BIBLIOGRAFÍA

- Sistemas de detección de intrusos (Seguridad en Unix y Redes) – Red IRIS
- El sistema de detección de intrusiones (IDS) – Grupo Ático 4 (ciberseguridad)
- Banco de Preguntas y Respuestas de redes, Categoría: CCNA CyberOps - ¿Qué es un sistema de detección de intrusiones basado en host (HIDS)?
- Documentación oficial Python 3 - https://www.python.org/doc/
- StackOverflow

Dependencias del proyecto:

- Python 3.x
- Hashlib