

# OSSEC (HIDS) Project

IT Infrastructure Security

Simone Giordano

<b>1. Introduction</b>	<b>1</b>
1.1. IDS (Intrusion Detection System)	1
<b>2. HIDS (Host Intrusion Detection System)</b>	<b>2</b>
<b>3. OSSEC HIDS Overview</b>	<b>3</b>
<b>4. OSSEC Hands-On</b>	<b>4</b>
4.1 Installation	4
4.2 GUI/WUI troubles	4
<b>5. OSSEC Use-Cases</b>	<b>4</b>
5.1 Active Response to an SSH brute force attempt	5
5.2 Alteration of file systems by third party programs	9
5.3 Detect and Block Shellshock attack	12
5.4 Rootkit Detection	14
<b>6. Conclusions</b>	<b>15</b>

# 1. Introduction

In today's Internet landscape, **security** is a priority.

Since security threats are extremely varied in nature (ranging from rootkits to DDoS attacks), there are many different tools to monitor, prevent, and mitigate them. In addition to other security mechanisms, such as firewalls and antivirus software, it is important to implement an **Intrusion Detection Systems (IDS)**.

## 1.1. IDS (Intrusion Detection System)

An **IDS is a device or software application** that monitors a network or systems for malicious activity or policy violations. Any detected activity or violation is typically reported either to an administrator or collected centrally to be audited. There is a wide spectrum of IDS, ranging from antivirus software to hierarchical systems that monitor the traffic of an entire backbone network. The most common classifications are **Network Intrusion Detection Systems (NIDS)** and **Host Intrusion Detection Systems (HIDS)**.

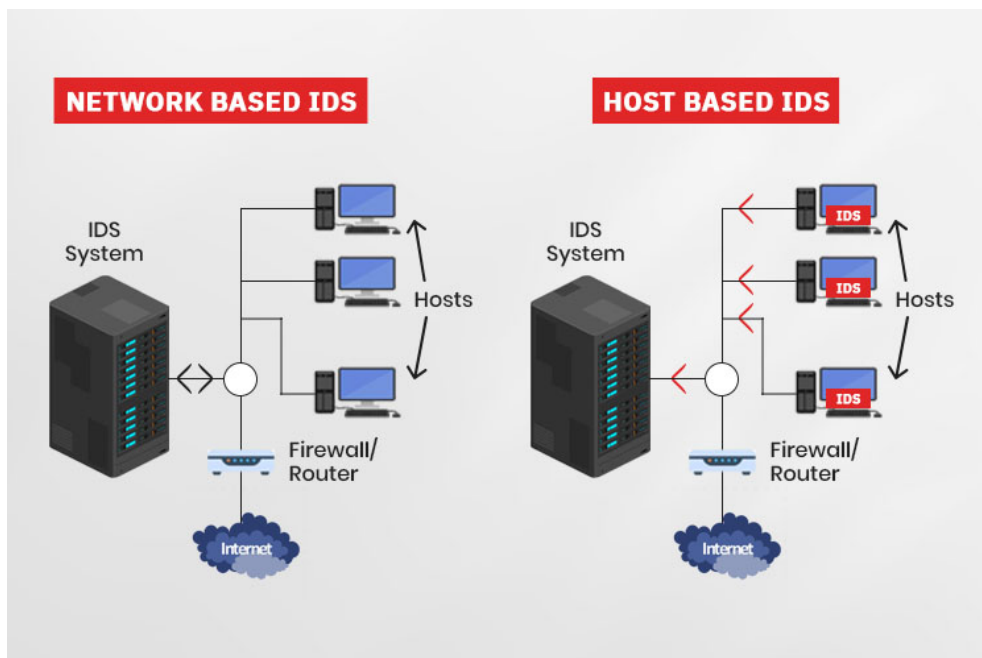


Image 1.1: NIDS vs HIDS from [Source](#)

## 2. HIDS (Host Intrusion Detection System)

To secure an infrastructure in these new modern scenarios, a good approach could be the deployment of an **Host Intrusion Detection System (HIDS)**.

HIDS deals with security threats inside host machines themselves (for instance privilege escalation, rootkits, logs, etc.), rather than on the network.

This has some clear advantages:

- **Configuration compatibility:** HIDS needs little additional configuration to be accommodated on existing infrastructure.
- **Performance:** HIDS-specific protocols tend to be lightweight, having little impact on the network and on the host.
- **Flexibility:** Since each host is individually monitored, adding/removing hosts and services is easier.
- **Granularity:** Each host can have many different monitoring rules, which means that we can be as specific as we want on what to monitor for security threats.

### 3. OSSEC HIDS Overview

OSSEC (Open Source Host-Based Intrusion Detection System) is an HIDS that monitors a wide assortment of **event** types that may indicate an invasion and matches these events to rules that, in turn, trigger responses.

These components match **rules** and trigger **warnings** and **responses**:

- **Rules**

Events in the system can be virtually anything;

OSSEC already comes with many built-in events, like:

port knocking, failed authentication, too many connections from a host, USB insertion, logs generation and content, etc.

Rules are used to **catch** these events on agents and **send** them to the manager, which will generate **warnings** and trigger **responses**. Rules are extremely customizable and extendable, and are categorized in levels, from 0 (least severe) to 15 (most severe).

- **Warnings**

Once a rule is triggered, it generates **warnings**. These warnings can then be forwarded via email or integrated with other software (like slack, instant messengers or custom scripts).

- **Responses**

After a rule is matched, then the monitor triggers the response on agents, such as denying a specific host, or stopping a given process, etc.

## 4. OSSEC Hands-On

### 4.1 Installation

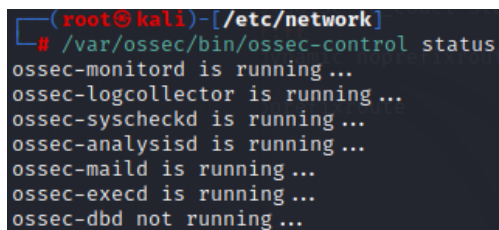
The installation of the OSSEC is pretty straight-forward, and is guided step by step in the setup tool.

OSSEC has been installed in the “local” configuration, s.t. it is similar to server and agent installation, except that the server has already been configured to listen for communication from the agents.

There is no need to install the in server/agent configuration, considering that there are not multiple hosts/agents to control.

### 4.2 GUI/WUI troubles

After installing the program, and checked the correct functioning of it with the command: `/var/ossec/bin/ossec-control status`

A terminal window screenshot showing the command `/var/ossec/bin/ossec-control status` being executed. The output lists the status of various OSSEC services: `ossec-monitor` is running, `ossec-logcollector` is running, `ossec-syscheckd` is running, `ossec-analysisd` is running, `ossec-maild` is running, `ossec-execd` is running, and `ossec-dbd` is not running.

```
(root@kali)-[/etc/network]
# /var/ossec/bin/ossec-control status
ossec-monitor is running...
ossec-logcollector is running...
ossec-syscheckd is running...
ossec-analysisd is running...
ossec-maild is running...
ossec-execd is running...
ossec-dbd not running...
```

There was the need to install a tool to visualize and analyze, almost in real time, the events caught by OSSEC. The most famous and used one was [ossec-wui](#), but it had a huge problem, that is deprecated and unmaintained from 3 years, this led to compatibility problems with my php version installed (it required php <= 5.4, but php 8 was installed).

To overcome this problem, I have created a python script that, every second, reads the content stored in `/var/ossec/logs/alerts/alerts.log` and, after picked up the relevant values, of every last log saved in the file, some functionalities are triggered, to make easier the readability of the events (more on this later).

## 5. OSSEC Use-Cases

### 5.1 Active Response to an SSH brute force attempt

The first use-case that will be presented is based on the **SSH brute force attack**.

According to the SANS Institute Security Risks Report for 2007, brute-force/dictionary attacks against remote services such as SSH, are one of the Top-20 most common forms of attack on the Internet that compromise servers, so a correct mitigation to this common attack is essential in modern hosts.

OSSEC could be a possible solution thanks to the **active-response** feature triggered by specific conditions or rules.

OSSEC comes with a preset of **rules** for many use-cases, from apache servers to mysql, and, of course, for sshd attacks. The rule that it is needed to mitigate the ssh brute force attack is the rule with id ="5712" in the file "sshd\_rules", stored in

`/var/ossec/rules/sshd_rules.xml`

```
<rule id="5712" level="10" frequency="3" timeframe="120" ignore="60">
  <if_matched_sid>5710</if_matched_sid>
  <description>SSHD brute force trying to get access to </description>
  <description>the system.</description>
  <same_source_ip />
  <group>authentication_failures,</group>
</rule>
```

Now, to apply the rule and making it active by ossec, we need to modify the configuration file of the program, stored in `/var/ossec/etc/ossec.conf`, adding the two following snippets:

```
<command>
  <name>firewall-drop</name>
  <executable>firewall-drop.sh</executable>
  <expect>srcip</expect>
  <timeout_allowed>yes</timeout_allowed>
</command>
```

`<command></command>` specifies **what** ossec must do when it is triggered, where, in this case, adds the ip of the host that is trying to connect via ssh multiple times with wrong passwords, in the drop list of the iptables, which are essentials.

```
<active-response>
  <command>firewall-drop</command>
  <location>local</location>
  <rules_id>5712</rules_id>
  <timeout>1800</timeout>
</active-response>
```

`<active-response></active-response>` specifies **when** ossec must be triggered and with what command, where in this case is the one written before. Obviously the rule, determined by a specific id, that must be applied is the one found before.

Before continuing, let's take a look to the iptables, which of course are empty with all ACCEPT:

```
(kali㉿kali)-[~]
$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Now we can go to the client-side and check the connection with our server (the one used until now),

```
(kali㉿kali)-[~]
$ ping 192.168.0.157
PING 192.168.0.157 (192.168.0.157) 56(84) bytes of data:
64 bytes from 192.168.0.157: icmp_seq=1 ttl=63 time=117 ms
64 bytes from 192.168.0.157: icmp_seq=2 ttl=63 time=3.29 ms
64 bytes from 192.168.0.157: icmp_seq=3 ttl=63 time=4.38 ms
64 bytes from 192.168.0.157: icmp_seq=4 ttl=63 time=3.40 ms
^C
— 192.168.0.157 ping statistics —
4 packets transmitted, 4 received, 0% packet loss, time 3007ms
rtt min/avg/max/mdev = 3.286/31.941/116.700/48.937 ms
```

and as we can see, the connection is correctly established and the PCs can speak to each other with a ping command.

Now let's try to connect multiple times in ssh, with the command

`"ssh kali@192.168.0.157"` inserting every time a wrong password, simulating a brute-force attack:

```
(root@kali)-[/home/kali]
# ssh kali@192.168.0.157
The authenticity of host '192.168.0.157 (192.168.0.157)' can't be established
.
ED25519 key fingerprint is SHA256:SIzDRAkgjNk1GcvUIRHXzFJztD6GGr78St3/hRsrOq8
.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.0.157' (ED25519) to the list of known hos
ts.
kali@192.168.0.157's password:
Permission denied, please try again.
kali@192.168.0.157's password:
Permission denied, please try again.
kali@192.168.0.157's password:
kali@192.168.0.157: Permission denied (publickey,password).

(root@kali)-[/home/kali]
# ping 192.168.0.157
PING 192.168.0.157 (192.168.0.157) 56(84) bytes of data.
```

After three attempts, the server automatically disconnects the client and, as we can see, it is not possible anymore to ping the server from the client, all while on the server side nothing seems to have happened.

On the server, where OSSEC is installed, it is possible to check what happened in the `/var/ossec/logs/alerts/alerts.log` file, where it is possible to see how many attempts the client has performed to connect via ssh and the source ip of the client for every attempt.

```
** Alert 1647876963.11187: - syslog,sshd,authentication_failed,
2022 Mar 21 11:36:03 kali→/var/log/auth.log
Rule: 5716 (level 5) → 'SSHD authentication failed.'
Src IP: 192.168.0.213
User: kali
Mar 21 11:36:03 kali sshd[170334]: Failed password for kali from 192.168.0.213 port 53063 ssh2

** Alert 1647876967.11479: - syslog,sshd,authentication_failed,
2022 Mar 21 11:36:07 kali→/var/log/auth.log
Rule: 5716 (level 5) → 'SSHD authentication failed.'
Src IP: 192.168.0.213
User: kali
Mar 21 11:36:06 kali sshd[170334]: Failed password for kali from 192.168.0.213 port 53063 ssh2

** Alert 1647876969.11771: mail - syslog,access_control,authentication_failed,
2022 Mar 21 11:36:09 kali→/var/log/auth.log
Rule: 2502 (level 10) → 'User missed the password more than one time'
Src IP: 192.168.0.213
User: kali
Mar 21 11:36:07 kali sshd[170334]: PAM 2 more authentication failures; logname= uid=0 euid=0 tty=ssh r
user= rhost=192.168.0.213 user=kali
```



But in this file there aren't any informations about a possible active response, as matter of fact, those informations are stored in `/var/ossec/logs/active-responses.log`, where, after the 3 attempts, the rules that the iptables needs to apply are shown.

```
GNU nano 6.2 active-responses.log
Mon Mar 21 11:36:09 AM EDT 2022 /var/ossec/active-response/bin/host-deny.sh add - 192.168.0.213 16478>
Mon Mar 21 11:36:09 AM EDT 2022 /var/ossec/active-response/bin/firewall-drop.sh add - 192.168.0.213 1>
```

To verify them, the `"iptables -L"` command is launched, and the result is the following, where all the packets coming from the source IP of the client, are dropped, denying possible more attempts to that specific client to connect via ssh to our server:

```
(root@kali)-[/var/ossec/rules]
# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination
DROP      all  --  192.168.0.213          anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

This example has shown how OSSEC can interrupt an attack without a user interaction, but just with a correct initial setup of the program.

## 5.2 Alteration of file systems by third party programs

OSSEC is widely used to generate **alerts** for the user, rather than create responses to possible threats. Those alerts can be delivered to the user via several ways, such as via mail and via integration of third party apps that read the log file and create **warnings**. To avoid the download of those apps, that are not always free and difficult to set-up, the python script mentioned before will be used to generate the warnings.

The File Integrity Monitoring (FIM) component of OSSEC detects and alerts when operating system or application files are modified. This capability is often used to detect access or changes to sensitive data.

In this case the `/etc/network/interfaces`, which is a file that is used to define static and dynamic IP addresses for the interfaces, setup routing information and default gateways, masquerading network bonding and more, will be changed.

To change it I have created a simple python script that will change that file and automatically will restart the network of the user, applying the modified ip address.

```
script_write_interfaces.py X
script_write_interfaces.py > ...
1 import subprocess
2 with open("/etc/network/interfaces", "a") as myfile:
3     myfile.write("\nauto eth0\niface eth0 inet static\naddress 192.168.0.70/24")
4
5 subprocess.call(['sudo /sbin/ifdown eth0 && sleep 10 && sudo /sbin/ifup --force eth0'], shell=True)
```

So this script will open the interfaces file and append in it the new ip address, after that, thanks to the subprocess function, the interface eth0 will be shutdown and then upped forcing the applying of the new ip address as seen in the “before” and “after” in the network lookup, thanks to the `ip a` command.

```
(root@kali)~[/etc/network]
# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
   link/ether 08:00:27:95:bd:54 brd ff:ff:ff:ff:ff:ff
   inet 192.168.0.157/24 brd 192.168.0.255 scope global dynamic noprefixroute eth0
       valid_lft 83785sec preferred_lft 83785sec
   inet6 fe80::a00:27ff:fe95:bd54/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

ip a before

```
(root@kali)-[/etc/network]
# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:95:bd:54 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.70/24 brd 192.168.0.255 scope global eth0
        valid_lft forever preferred_lft forever
```

ip a after

In the meanwhile, the syscheck process of OSSEC was listening to all the changes in the specified directories in the ossec.conf file, and, with the added commands, which are “report\_change = yes” and “realtime=yes”, OSSEC will immediately create an alert and will not wait the frequency of detection if it detects a change:

```
<syscheck>
<!-- Frequency that syscheck is executed - default to every 22 hours -->
<frequency>550</frequency>
<auto_ignore>no</auto_ignore>
<!-- Directories to check (perform all possible verifications) as written now, it is in real time edit
<directories report_changes="yes" realtime="yes" check_all="yes">/etc,/usr/bin,/usr/sbin</directories>
<directories report_changes="yes" realtime="yes" check_all="yes">/bin,/sbin</directories>
```

As matter of fact, after few seconds, an alert has been generated, stating that a change in the /etc/network/interface has been performed with criticity level:7

```
** Alert 1648656284.203595: mail - ossec,syscheck,
2022 Mar 30 12:04:44 kali->syscheck
Rule: 550 (level 7) -> 'Integrity checksum changed.'
Integrity checksum changed for: '/etc/network/interfaces'
Size changed from '240' to '300'
Old md5sum was: 'fc5bcd80dade50331031ce031084fde5'
New md5sum is : '1e0a6c57da2508a1426c18f07c9d4c93'
Old sha1sum was: '0a01e89f996f859b5184c3509ff513d60ff3636c'
New sha1sum is : 'fa8a06c00e4c173399d55b7de6c5894dbd275db2'
```

and, with particular relevance, OSSEC creates an alert also for the performed ifdown and ifup commands done by the python script adding more informations.

```
** Alert 1648658536.212576: - syslog,sudo
2022 Mar 30 12:42:16 kali->/var/log/auth.log
Rule: 5402 (level 3) -> 'Successful sudo to ROOT executed'
User: root
Mar 30 12:42:15 kali sudo:      root : TTY=pts/5 ; PWD=/home/kali/Documents/It_Infr_Sec ; USER=root ; COMMAND=/sbin/ifdown eth0

** Alert 1648658536.212861: - pam,syslog,authentication_success,
2022 Mar 30 12:42:16 kali->/var/log/auth.log
Rule: 5501 (level 3) -> 'Login session opened.'
Mar 30 12:42:15 kali sudo: pam_unix(sudo:session): session opened for user root(uid=0) by kali(uid=0)

** Alert 1648658536.213122: - pam,syslog,
2022 Mar 30 12:42:16 kali->/var/log/auth.log
Rule: 5502 (level 3) -> 'Login session closed.'
Mar 30 12:42:15 kali sudo: pam_unix(sudo:session): session closed for user root

** Alert 1648658546.213338: - syslog,sudo
2022 Mar 30 12:42:26 kali->/var/log/auth.log
Rule: 5402 (level 3) -> 'Successful sudo to ROOT executed'
User: root
Mar 30 12:42:25 kali sudo:      root : TTY=pts/5 ; PWD=/home/kali/Documents/It_Infr_Sec ; USER=root ; COMMAND=/sbin/ifup --force eth0
```

The script that reads the log file and dynamically shows the alerts created by OSSEC is the following:

```
import os
import time

def read_file():
    with open("/var/ossec/logs/alerts/alerts.log", "r") as f:
        fileLines = [x.strip() for x in f.readlines()]
    return fileLines

initial = read_file()
while True:
    current = read_file()
    if initial != current:
        for index, line in enumerate(current):
            if line not in initial:
                id_alert = line
                rule_line = current[index+2]
                user = current[index+3]
                useful_info = current[index+4]
                for i, word in enumerate(rule_line.split()):
                    rule = rule_line.split()[i+1]

                if rule != 550:
                    print("Id alert:", id_alert)
                    print(rule_line)
                    print(user)
                    print("Useful infos:", useful_info)
                else:
                    print("Id alert:", id_alert)
                    print(rule_line)
                    print(user)
                    print(useful_info)
                    print(current[index+5])
```

## 5.3 Detect and Block Shellshock attack

Another interesting experiment is the detection and the blocking of an attempt of a **Shellshock attack**.

A Shellshock attack is the ability to inject shell commands via maliciously crafted web requests sent to Linux web servers. The pattern in such web requests is quite distinctive, and any instance of your servers being probed with Shellshock requests are fairly strong indicators of malicious probing worthy of automated countermeasures.

The chosen web server is “nginx”, and, after correct installation and start up, a crafted malicious snippet is sent to it, that is the following:

```
curl --insecure $ShellshockTarget -H "User-Agent: () { :; };  
/bin/cat /etc/passwd"
```

Where \$Shellshocktarget was previously instantiated to the local ip of the server.

After the execution of the command, it is possible to check the alert generated by ossec, always in the alerts.log file:

```
** Alert 1648824382.698533: mail - web,accesslog,attack,pci_dss_11.4,gdpr_IV_35.7.d,nist_800_53_SI.4,  
2022 Apr 01 10:46:22 kali→/var/log/nginx/access.log  
Rule: 31166 (level 15) → 'Shellshock attack attempt'  
Src IP: 127.0.0.1  
127.0.0.1 - - [01/Apr/2022:10:46:21 -0400] "GET / HTTP/1.1" 200 612 "-" "() { :; }; /bin/cat /etc/pass  
wd"
```

This alert has been generated, thanks to the fact that OSSEC, by default, imports all the logs generated by nginx (access.log and error.log in /var/log/nginx), and, with the **rule 31166**, OSSEC can identify the pattern of a shell shock attack and generate the correspondent alert.

```
<rule id="31166" level="15">  
  <if_sid>31101,31108</if_sid>  
  <regex>"\(\)\s*{\s*:\s*}\s*;|\(\)\s*{\s*foo:\s*}\s*;|'\(\)\s*{\s*ignored;\s*}\s*|'\(\)\s*{\s*g>  
  <description>Shellshock attack attempt</description>  
  <info type="cve">CVE-2014-6271</info>  
  <info type="link">https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271</info>  
  <group>attack,pci_dss_11.4,gdpr_IV_35.7.d,nist_800_53_SI.4,</group>  
</rule>
```

After the correct detection of the danger, OSSEC gives the opportunity to block the source ip of the host that crafted the malicious request to the server. The methodology is analogous to the one explained before, with the only difference of the rule stored in ossec.conf related to the active response rule that is the following:

```
<active-response>
  <disabled>no</disabled>
  <command>firewall-drop</command>
  <location>local</location>
  <rules_id>31166</rules_id>
  <timeout>300</timeout>
</active-response>
```

Blocking the ip with the iptables in linux

## 5.4 Rootkit Detection

OSSEC can be really useful also for the detection of **rootkit** programs in the system. Unfortunately I couldn't be able to have a working rootkit on my machine because of incompatibilities with the Linux kernel between the latter and the rootkit programs found online.

To overcome this problem, I thought to simulate the presence of a rootkit in the system, by creating a hidden file in the `/dev` folder which should only contain device-specific files. Any additional files, outside of the expected device-specific files, should be inspected because many rootkits use `/dev` as a storage partition to hide files.

So first of all I checked in the configuration file of OSSEC, `ossec.conf`, if the detection of rootkit was enabled, i.e, if the rule `<rootkit_files>` and `<rootkit_trojans>` were in the configuration, and it was so:

```
<rootcheck>
  <frequency>50</frequency>
  <rootkit_files>/var/ossec/etc/shared/rootkit_files.txt</rootkit_files>
  <rootkit_trojans>/var/ossec/etc/shared/rootkit_trojans.txt</rootkit_trojans>
  <system_audit>/var/ossec/etc/shared/system_audit_rcl.txt</system_audit>
  <system_audit>/var/ossec/etc/shared/cis_debian_linux_rcl.txt</system_audit>
  <system_audit>/var/ossec/etc/shared/cis_rhel_linux_rcl.txt</system_audit>
  <system_audit>/var/ossec/etc/shared/cis_rhel5_linux_rcl.txt</system_audit>
</rootcheck>
```

After decreasing the frequency of the check, to every 50 seconds (by default is every 2 hours), I created the hidden file in the `/dev` folder

```
(root@kali)-[/dev]
# touch .hid
```

And checked if the file was correctly in the `/dev` folder.

```
(root@kali)-[/var/ossec/etc]
# ls -a /dev | grep '^\.'
```

..  
.  
.hid

After almost 50 seconds, OSSEC correctly **detected** the hidden file in the `/dev` folder, and raised an alert, marking it as a possible rootkit.

```
** Alert 1649168067.6402: mail - ossec,rootcheck,
2022 Apr 05 10:14:27 kali→rootcheck
Rule: 510 (level 7) → 'Host-based anomaly detection event (rootcheck).'
```

File '/dev/.hid' present on /dev. Possible hidden file.

## 6. Conclusions

As we have seen in the previous chapters, an HIDS, and in particular, the OSSEC program, can have a huge impact on the security of a device, after a correct initial setup.

The possibility to integrate it with other programs, such as Splunk and Wazuh, augmenting the potential to discover new threats, is the main backlash on the use of the IDS, not anymore basing the creation of alerts on a signature base but also on a behavioral base, via, for example, machine learning algorithms.

The easy configuration of OSSEC and the pre-configured basic safety rules that the program delivers is one of the most important merits that OSSEC got, but, on the other hand, the lack of, also basic, Graphic user interface and a little flexibility in the changes of the rules in which the program works, such as the impossibility to block a single port of an host and not the entirety of it, blocking its ip address, leads to the impossibility for the program to be used in an enterprise context, but mostly in a didactical or domestic one.