Final Project

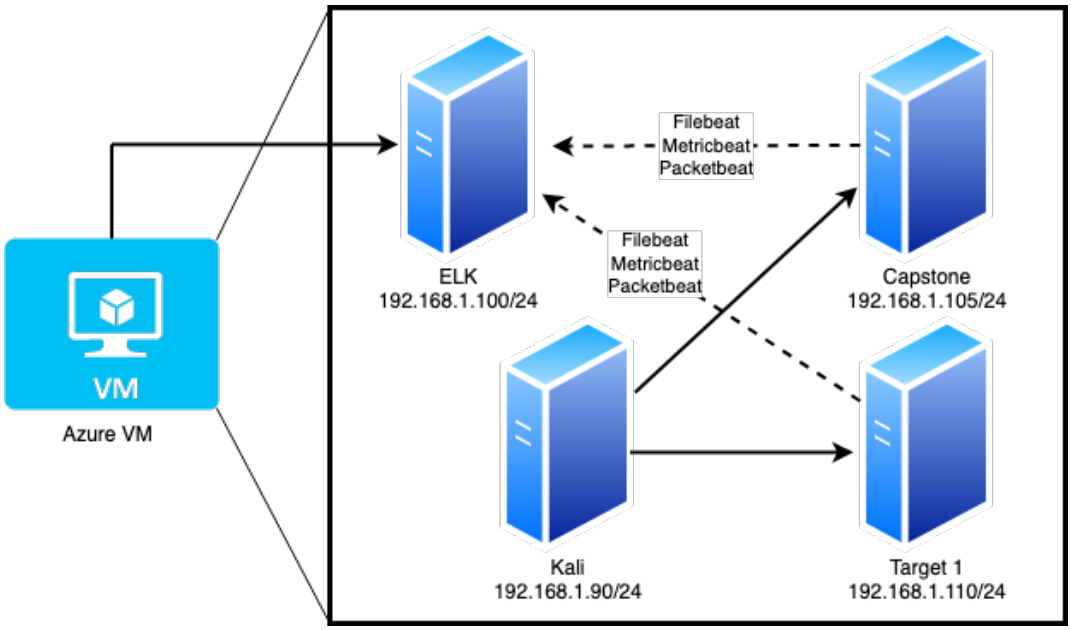
Brian Ferguson

8/18/2021

UT Austin Cybersecurity Bootcamp

Table of Contents

Network Topology

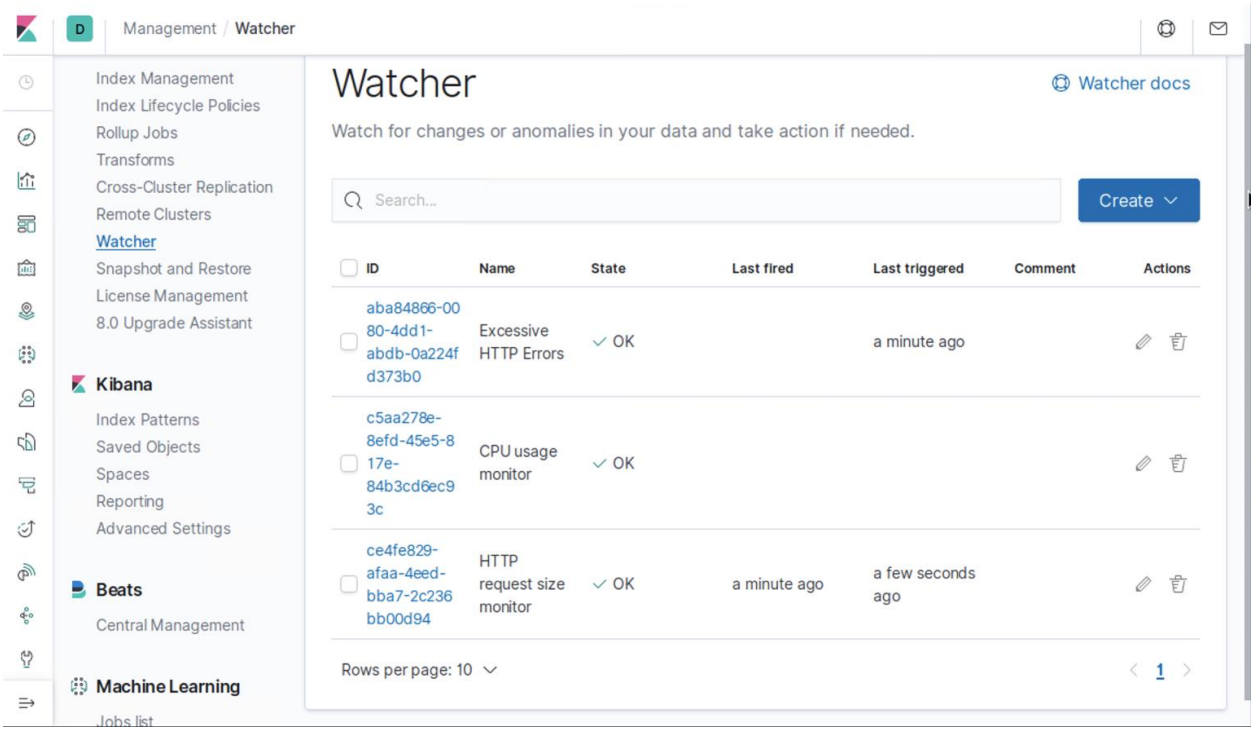


Virtual Machine	IP Address	Description
Kali	192.168.1.90	Kali Linux machine that will be used as an attacking vector toward the other machines in this operation
Capstone	192.168.1.105	Capstone is a vulnerable machine that we'll be using to test alerts
ELK Server	192.168.1.100	This server features an ELK installation with proper channels to monitor logs from both Capstone and Target 1 via Kibana
Target 1	192.168.1.110	This vulnerable VM has an installation of Wordpress on it, which we will be attacking with our Kali machine

Data Logs and Monitoring of Targets

Three types of alerts have been set up within Kibana to notify the Blue Team of the following potentially suspicious metrics:

- Excessive HTTP Errors
- CPU Usage Monitor
- HTTP Request Size Monitor



These logs are generated based on the data statistics of both Capstone and Target 1.

Alert 1: Excessive HTTP Errors

This alert indicates if the top five HTTP codes have been errors within the 400 range for over five minutes. Attackers can often gain useful information about a target by analyzing the errors it returns, and in such a case we have the ability to monitor their source IP and other activities.

To mitigate this potential threat, we should make modifications to our firewall settings which will help stop the offending traffic and/or implement an Intrusion Prevention System, which will automatically analyze the offending traffic based on a set of rules to identify and act on it accordingly.

Given the fact that this type of alert could be detecting either malicious or non-malicious activity, its reliability can't be considered extremely high, as it could easily generate both false positives *and* false negatives.

Alert 2: CPU Usage Monitor

This alert indicates unusual amounts of CPU usage on the server, which could be an indication of a Denial of Service attack by a malicious actor. The result of an excessively high CPU usage is an outage of server and site availability, which would likely impact both sales and customer satisfaction.

Although malware is sometimes used for this purpose of overloading the system, it's not always useful to an attacker when their goal is to compromise the target by stealing sensitive information, which can be manipulated or removed from the company's servers without any need for drawing unwanted attention by means of such high CPU usage activities. As such, this alert can be useful and reliable for attacks with a specific scope of intention, but its reliability is also limited by that scope.

Alert 3: HTTP Request Size Monitor

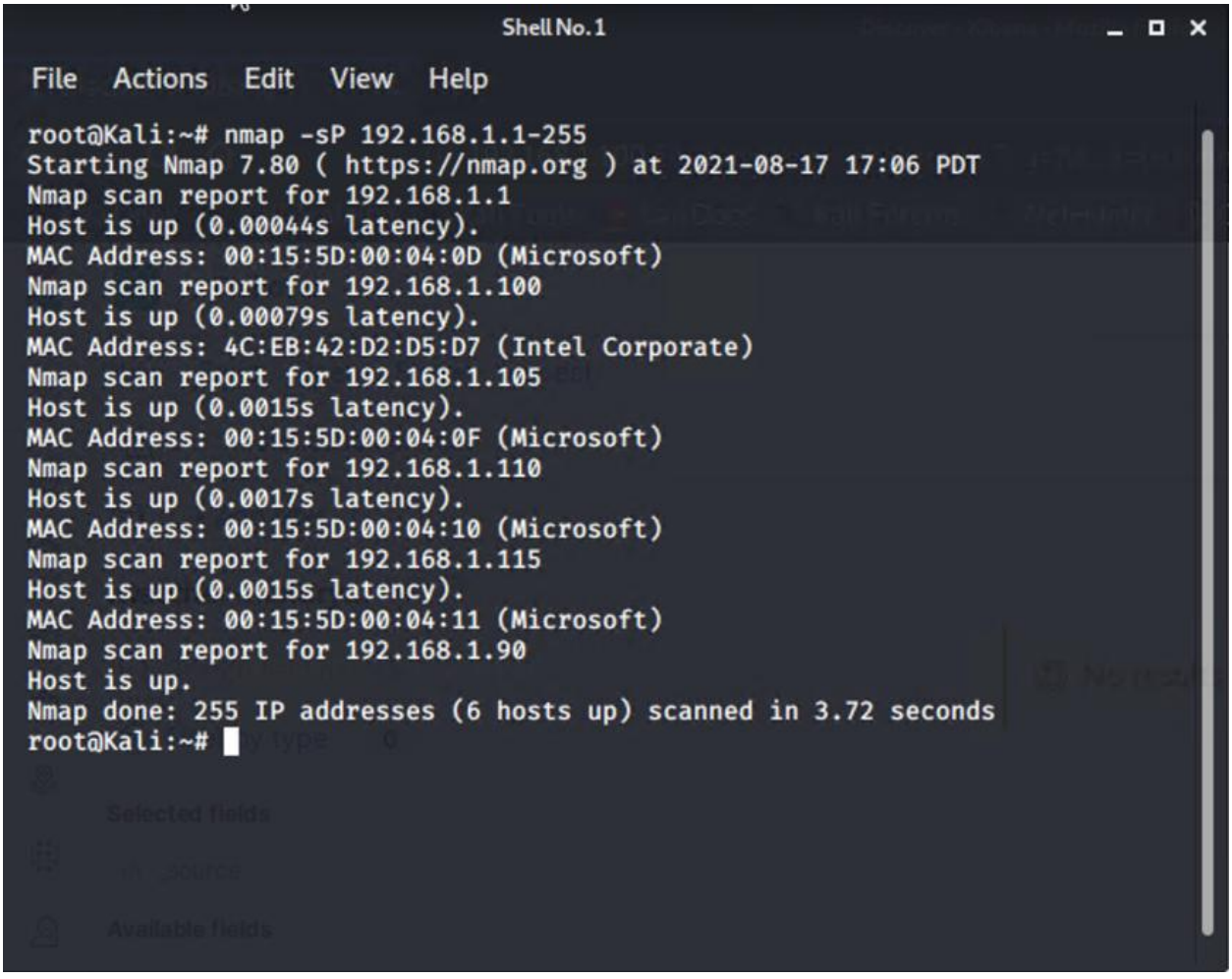
This alert notifies us if the HTTP request size goes over 3.5kb for more than one minute, which can be a sign of a malicious attack, such as a critical buffer overflow caused by remote code injection. Although these types of injections don't always result in a buffer overflow, they *can* result in a Denial of Service and therefore an interruption in site availability.

To help mitigate these problems, we'll need to implement a form of input validation and possibly input sanitization. Input validation makes sure that the information meets the proper criteria, and input sanitization actually *modifies* the input data to make sure that it is handled safely.

Overall, the reliability of this alert is fairly high, given that in most cases it indicates some type of harmful intent orchestrated by a malicious actor.

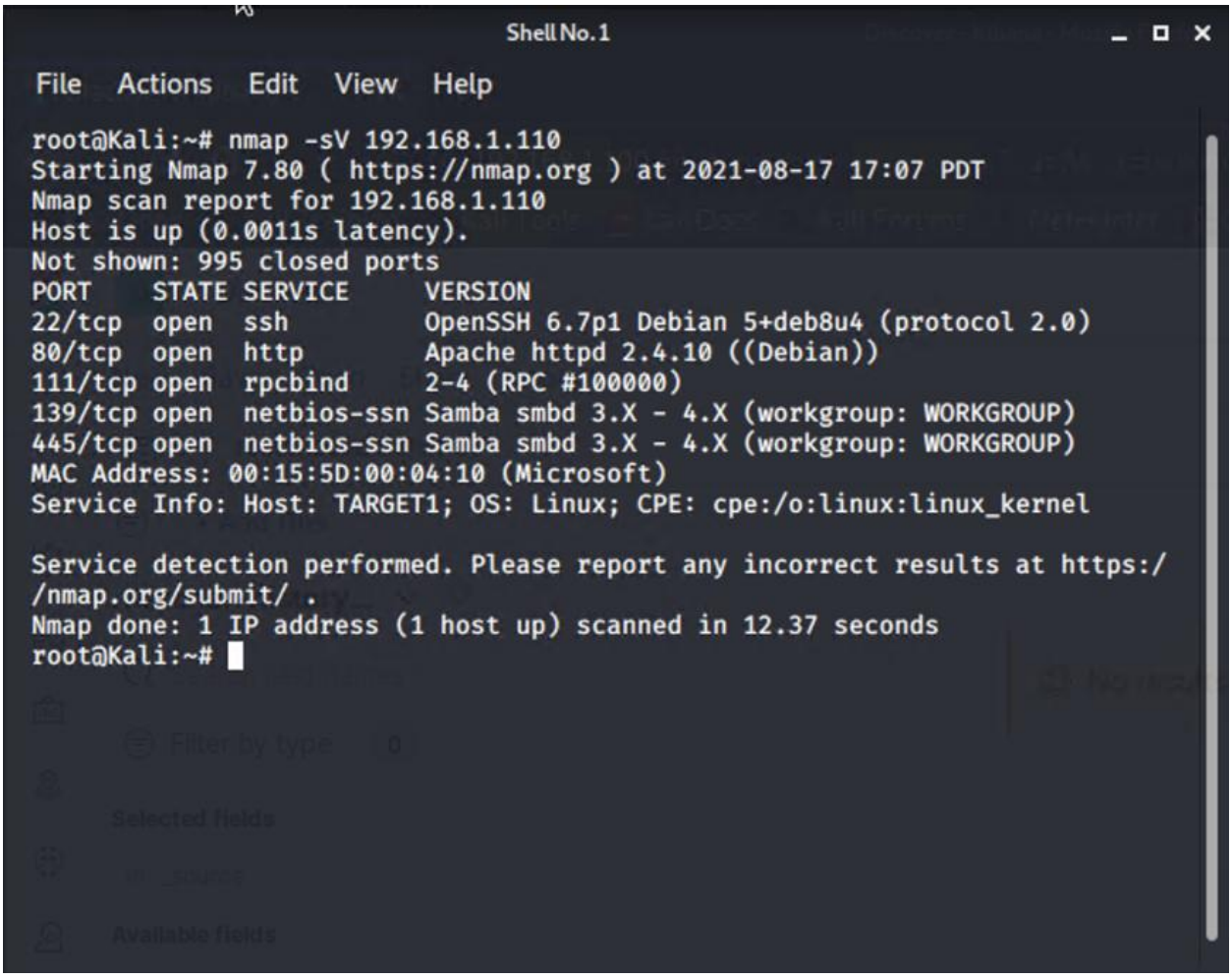
Exposed Services

By performing a ping sweep using nmap, I was able to discover several exposed IP addresses within the given range, starting with the gateway.
Command line: nmap -sP 192.168.1.1-255



Having prior knowledge of the target machine, I knew that 192.168.1.110 was the IP address for Target 1, so I executed an nmap command with a version sweep included in it. For this exercise, we'll be focusing on ports 80 and 22 (HTTP and SSH).

Command line: nmap -sV 192.168.1.110



Critical Vulnerabilities

After discovering the version of Apache (v 2.4.10) that this server is running, I was able to find the following critical vulnerabilities for this system:

1.) SECURITY: CVE-2013-5704 (cve.mitre.org) core: HTTP trailers could be used to replace HTTP headers late during request processing, potentially undoing or otherwise confusing modules that examined or modified request headers earlier. Adds "MergeTrailers" directive to restore legacy behavior.

2.) SECURITY: CVE-2014-3581 (cve.mitre.org) mod_cache: Avoid a crash when Content-Type has an empty value. PR 56924.

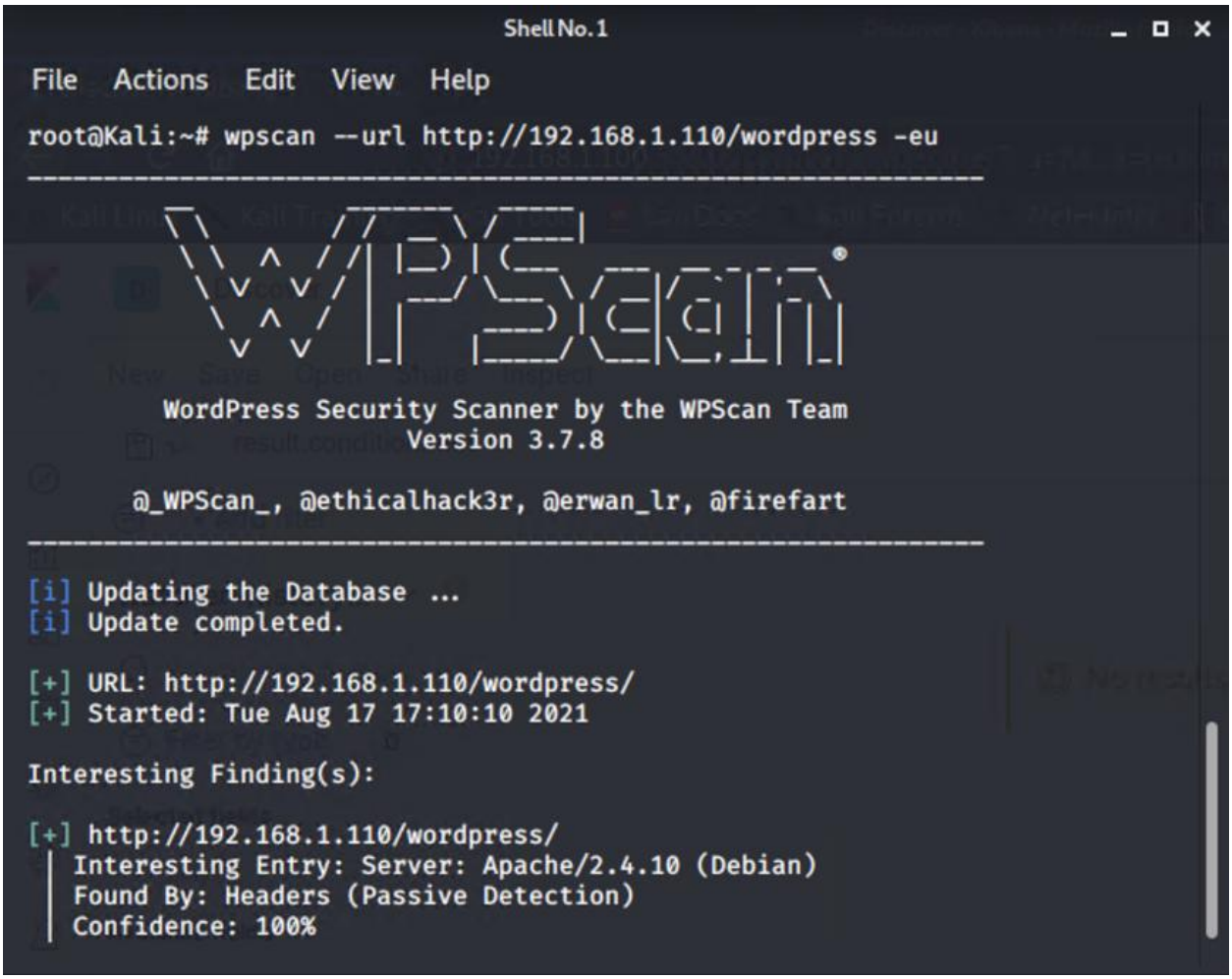
3.) SECURITY: CVE-2014-3583 (cve.mitre.org) mod_proxy_fcgi: Fix a potential crash due to buffer over-read, with response headers' size above 8K.

4.) SECURITY: CVE-2014-8109 (cve.mitre.org) mod_lua: Fix handling of the Require line when a LuaAuthzProvider is used in multiple Require directives with different arguments. PR57204.

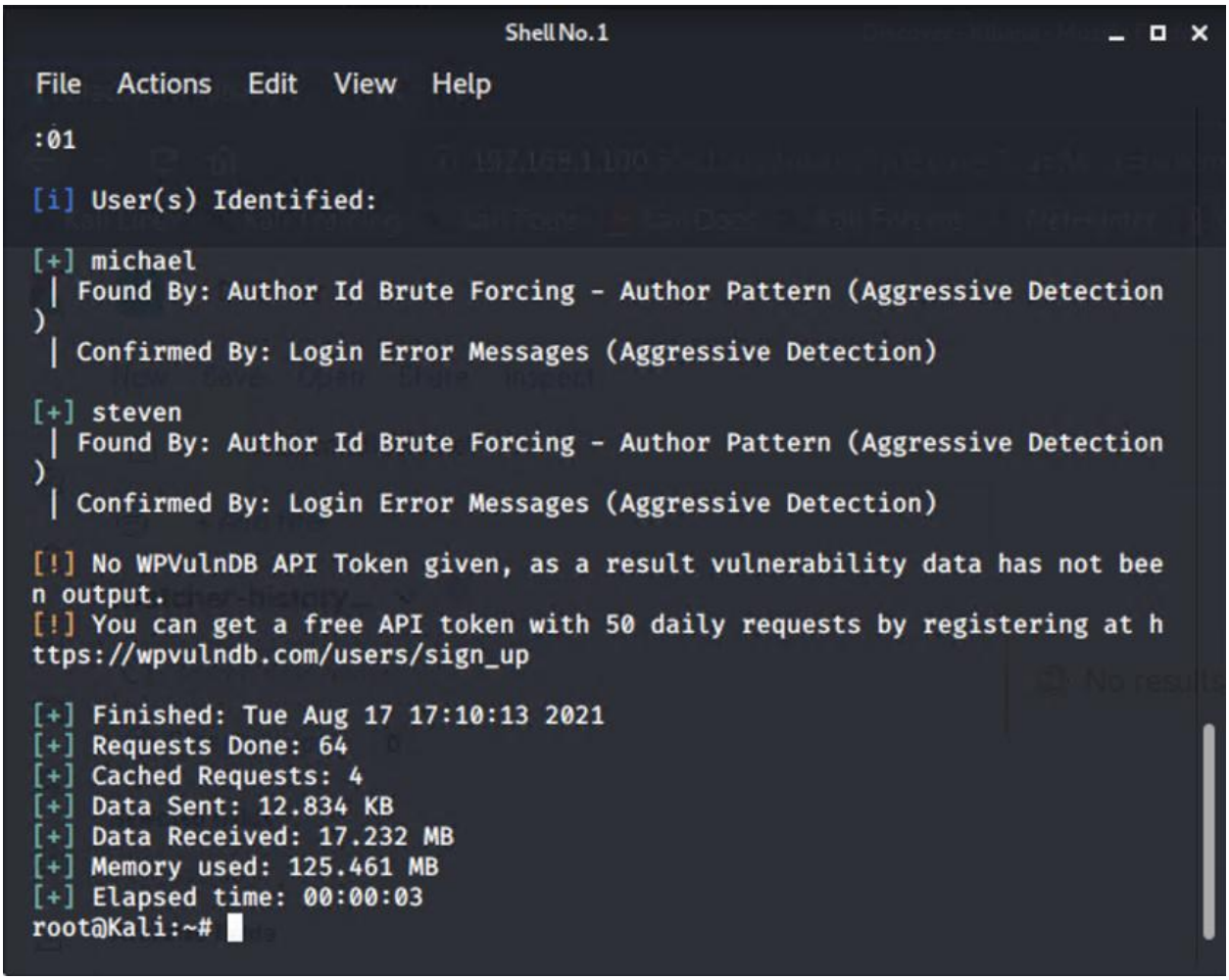
Exploitation and Walkthrough

Step 1 - Given that port 80 is open, it's likely that this port is hosting a website. Since Wordpress installations are one of the most common types of websites, I performed a WPScan on the target, which proved that to be the case.

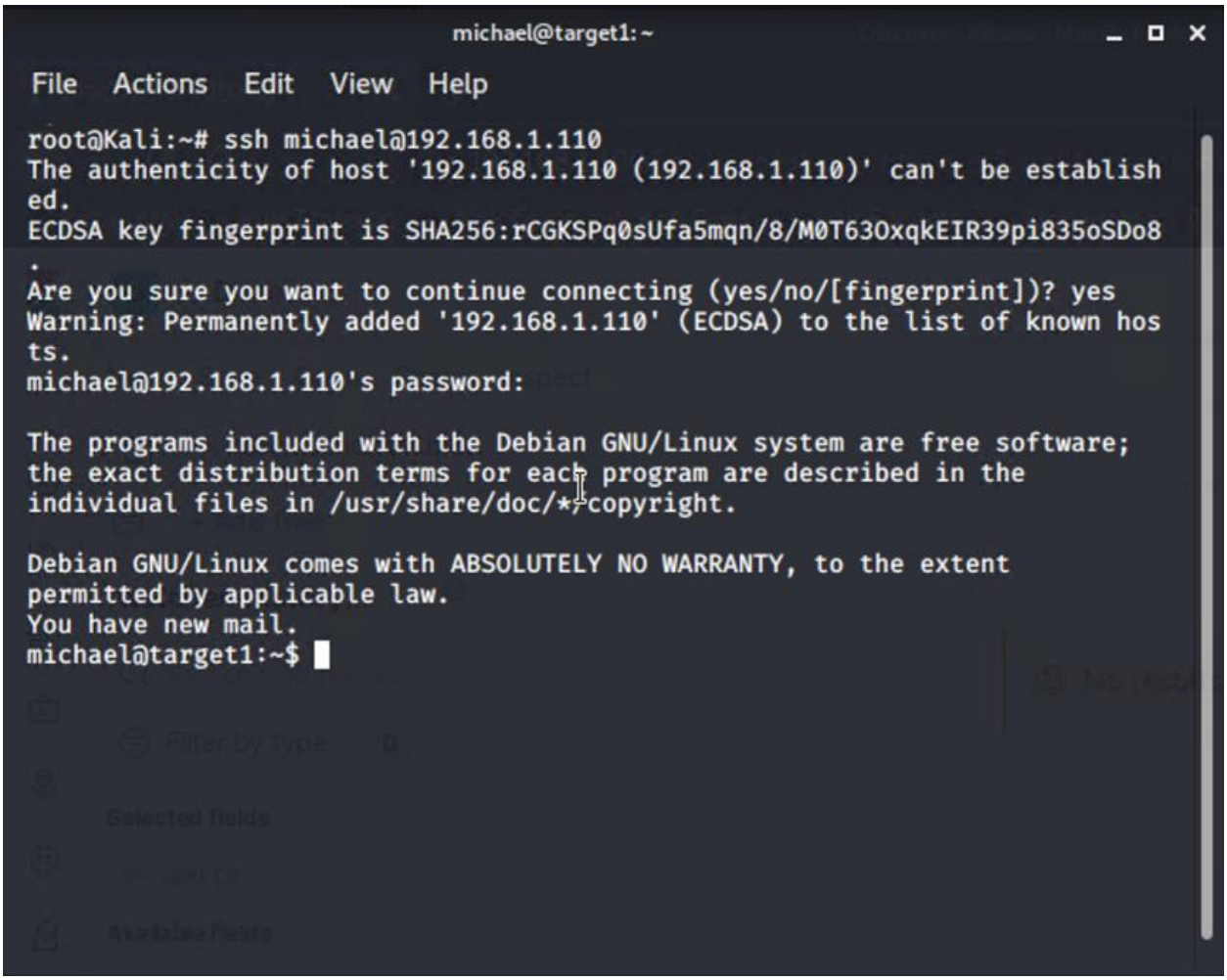
Command line: wpscan --url http://192.168.1.110/wordpress -eu



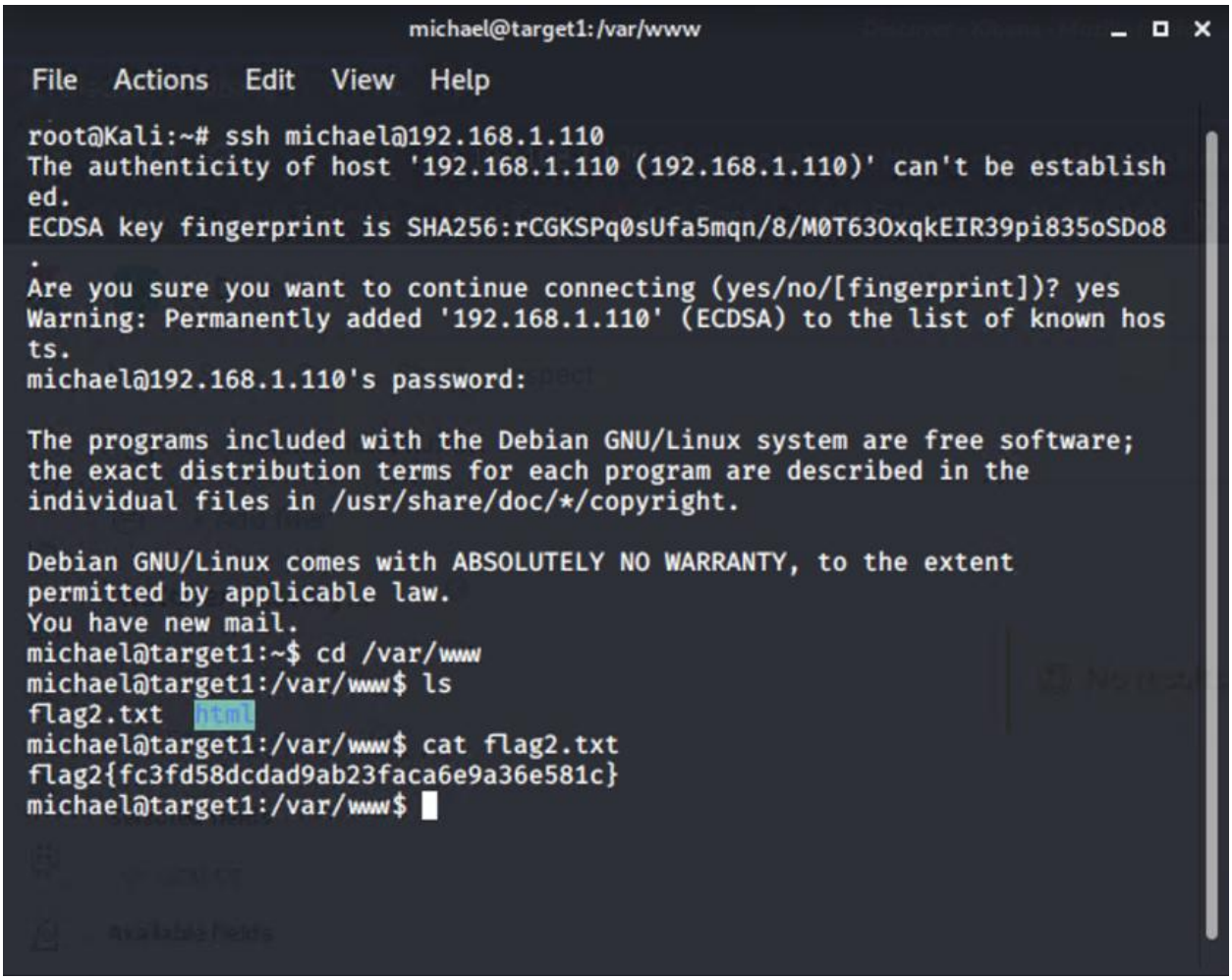
Step 2 - The results of this WPScan revealed two users on the site: a user named “michael” and another user named “steven”.



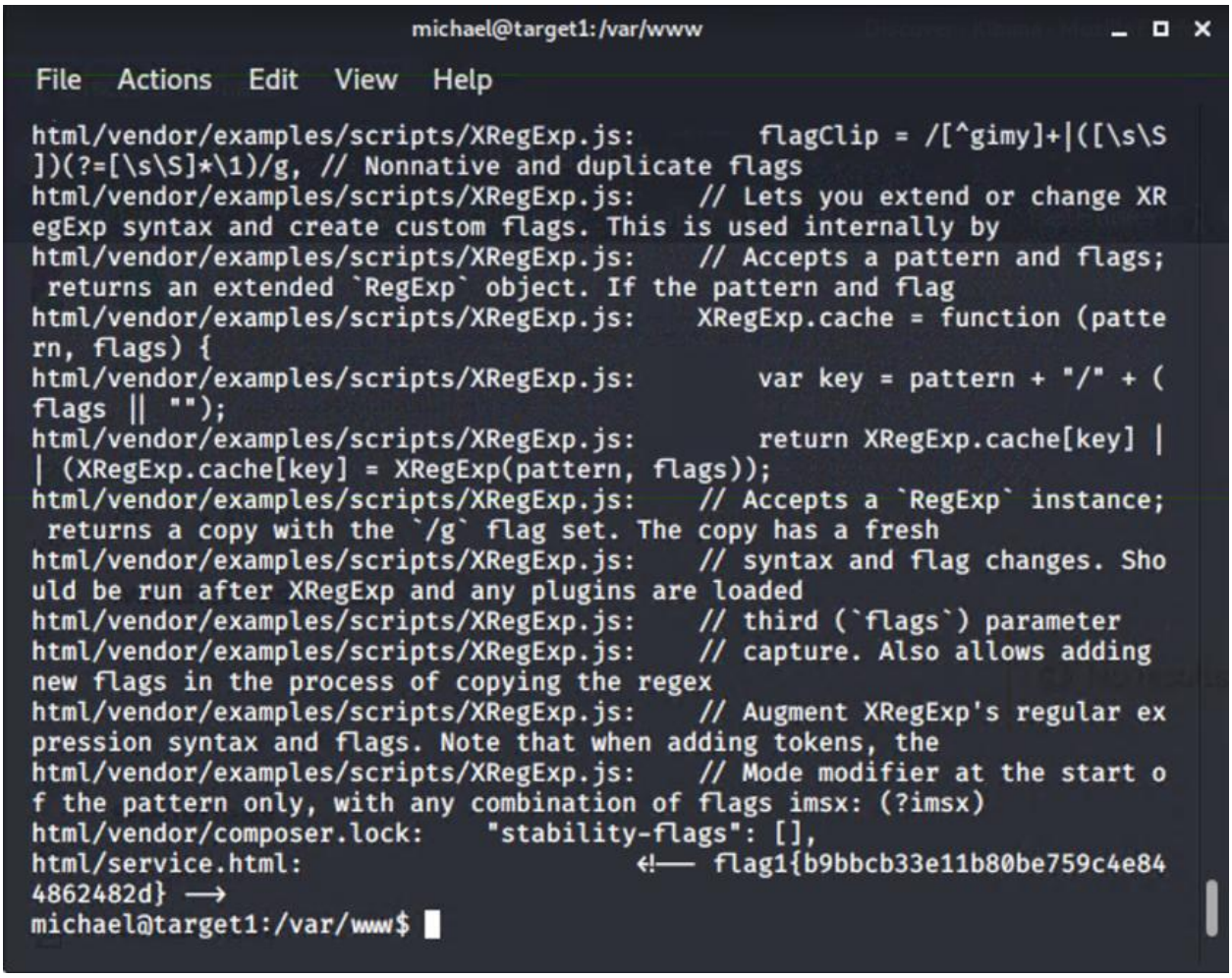
Step 3 - I was able to guess Michael's password (which was his own name) and log in to his account via SSH on port 22. Obviously, this type of password is inexcusable from a security standpoint.



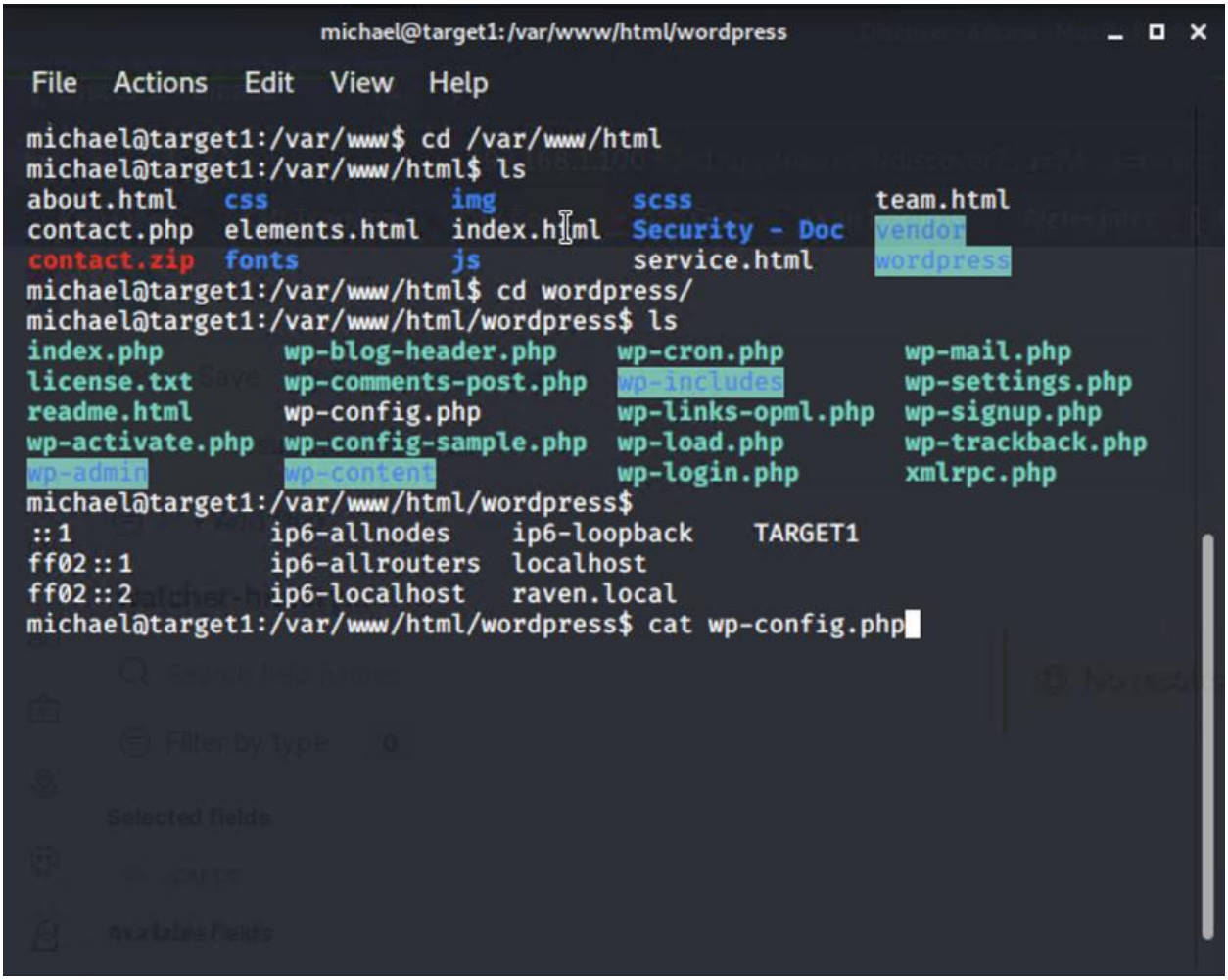
Step 4 - From this point, I began a search for sensitive information (which in this case is a series of flags to be captured), and found the first one in the /var/www directory.



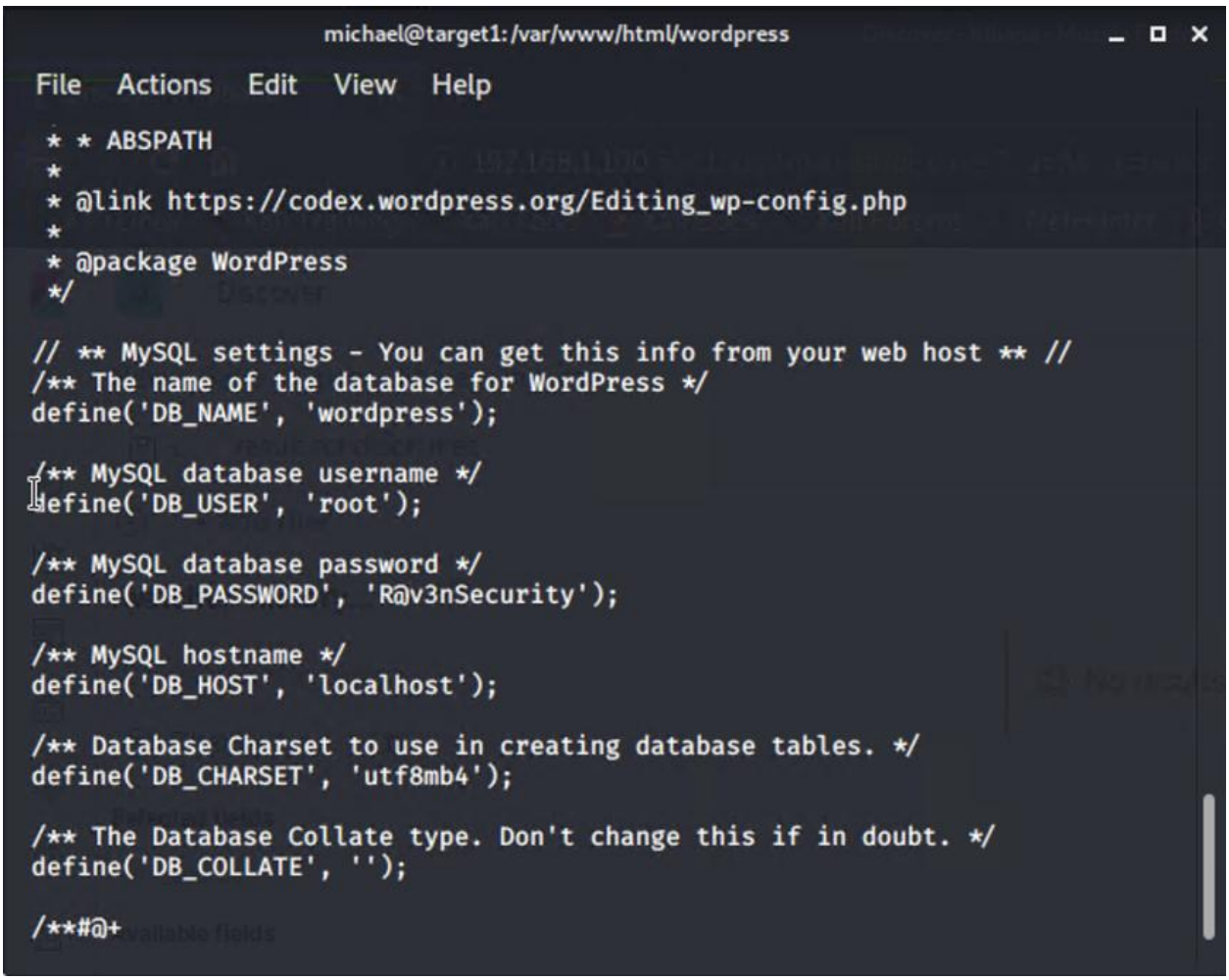
Step 5 - From here, we can use the grep command to search the html directory for the other flags. The results of the search show that the second flag is at the bottom of the list.
Command line: grep -RE flag html



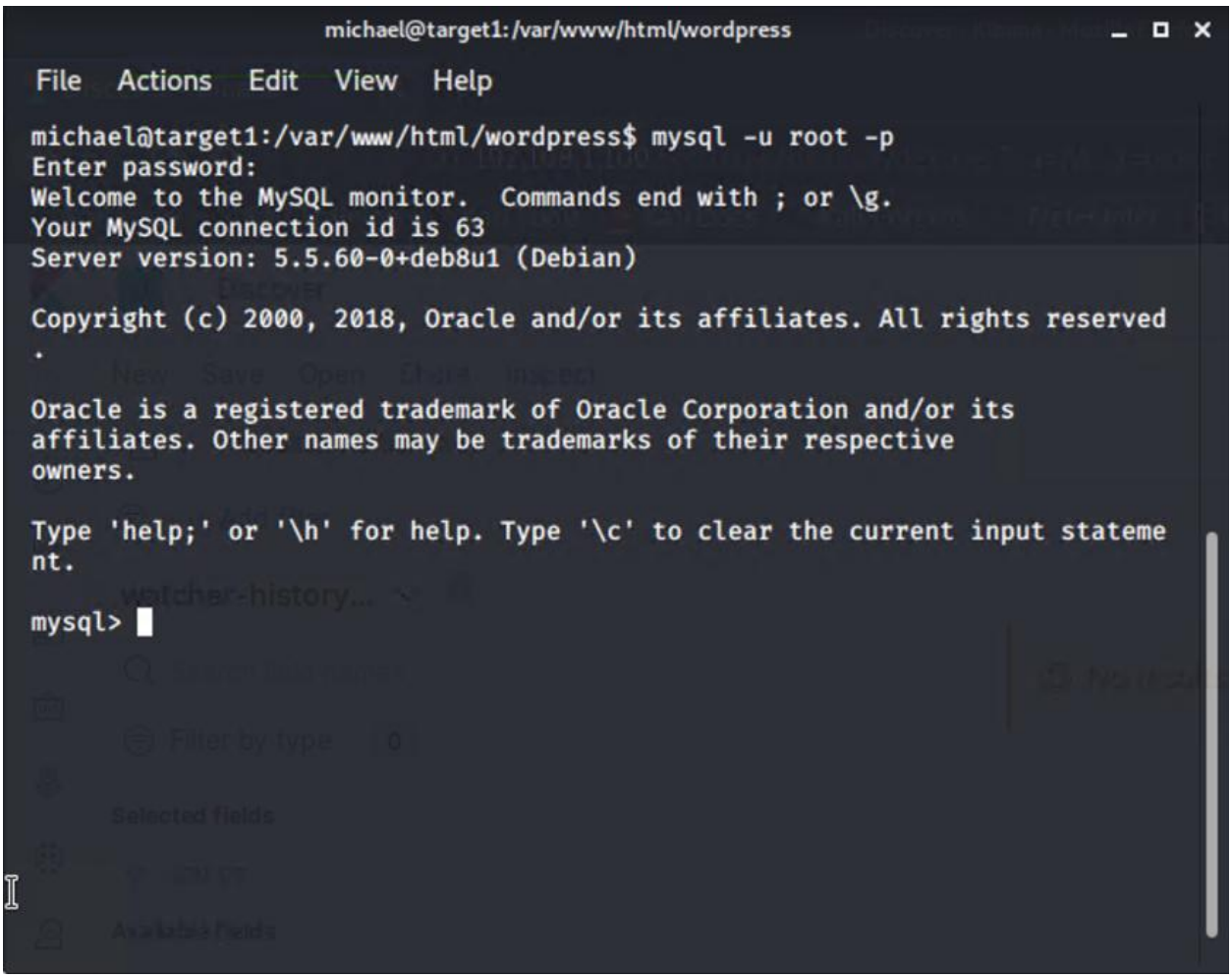
Step 6 - The next step is to check out the wordpress directory that's located within the html directory. Inside, we find the *wp-config.php* file, which stores the MySQL passwords for the site.



Step 7 - After using the cat command to display the contents of the *wp-config.php* file, we discover that the database password is 'R@v3nSecurity'.

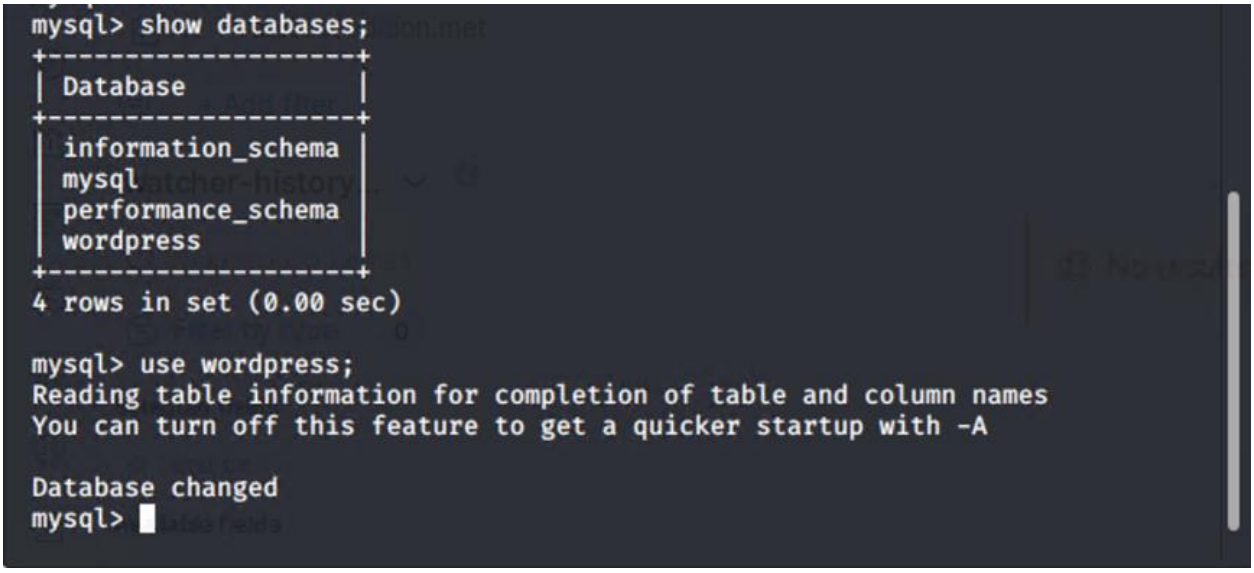


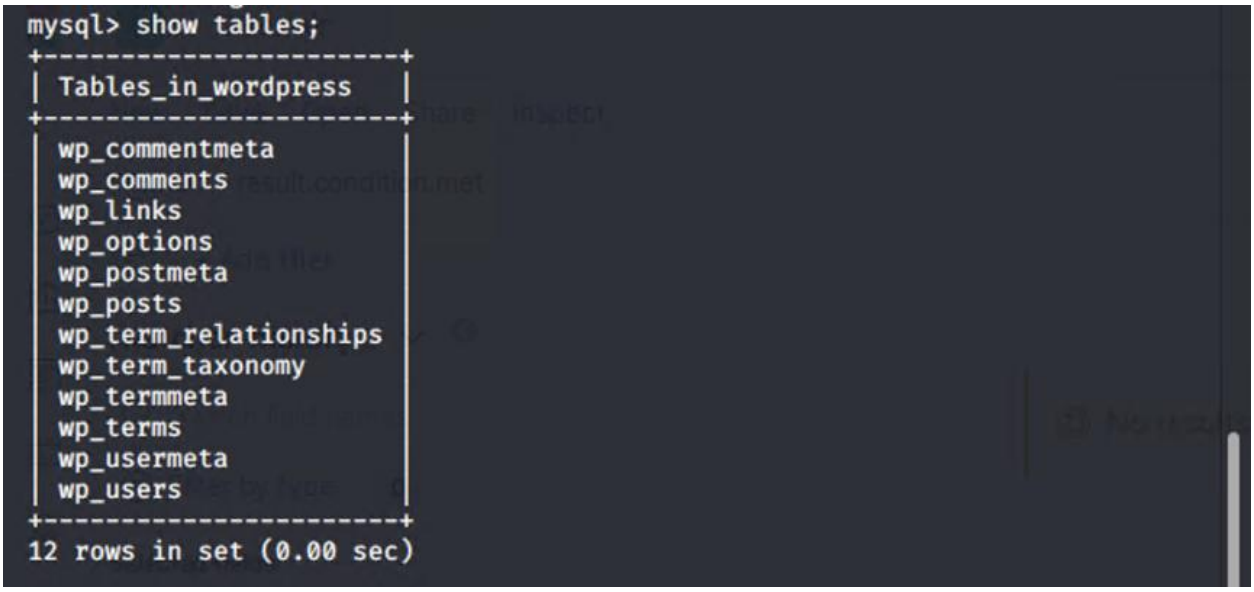
Step 8 - From here, we use this information to log in to the MySQL database and take a look at what we can find there.
Command line: mysql -u root -p



Step 9 - Now that we're inside the database, we enumerate the databases and then move onward to access the *wp_posts* table.

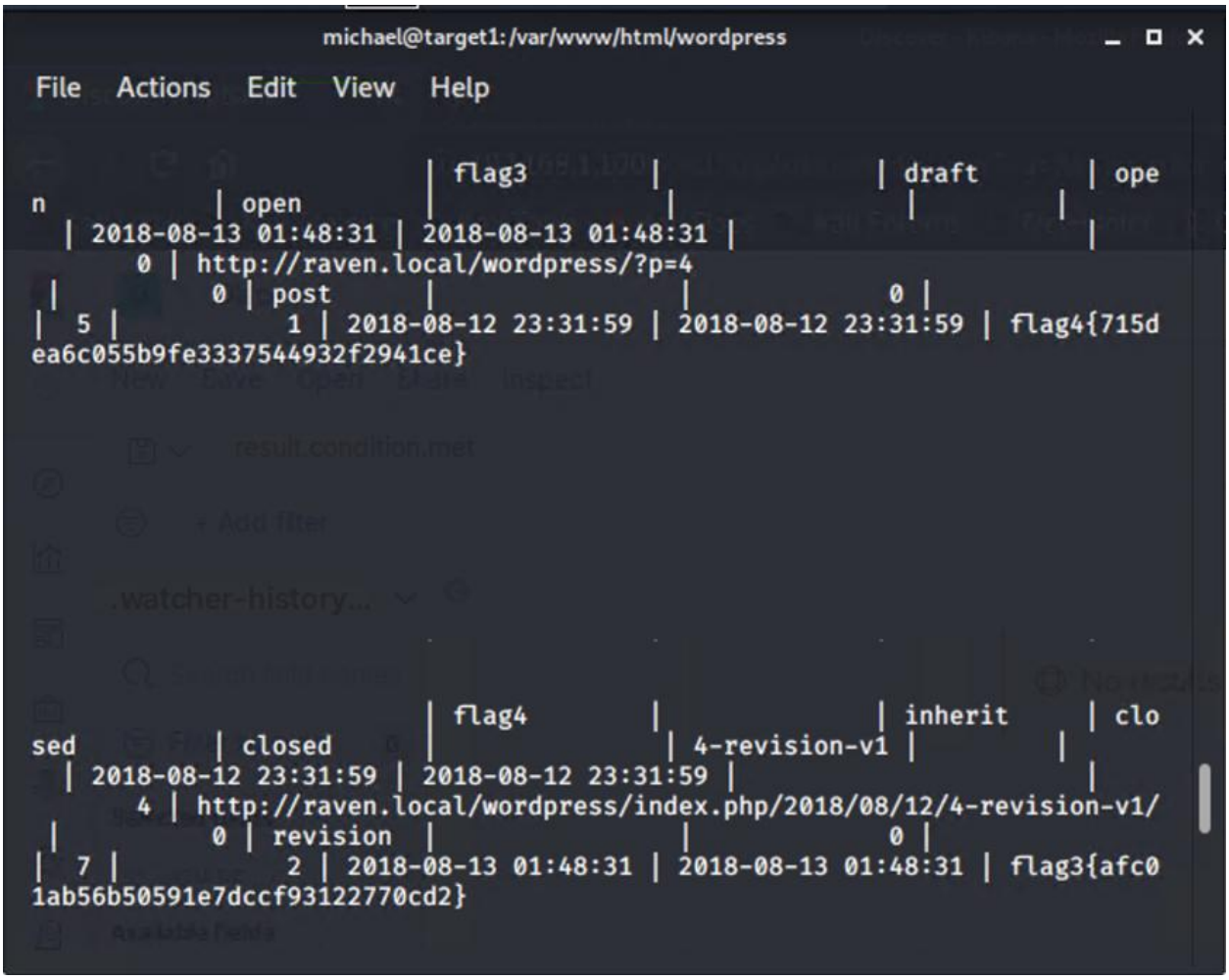
Command line: show databases;
use wordpress;
show tables;





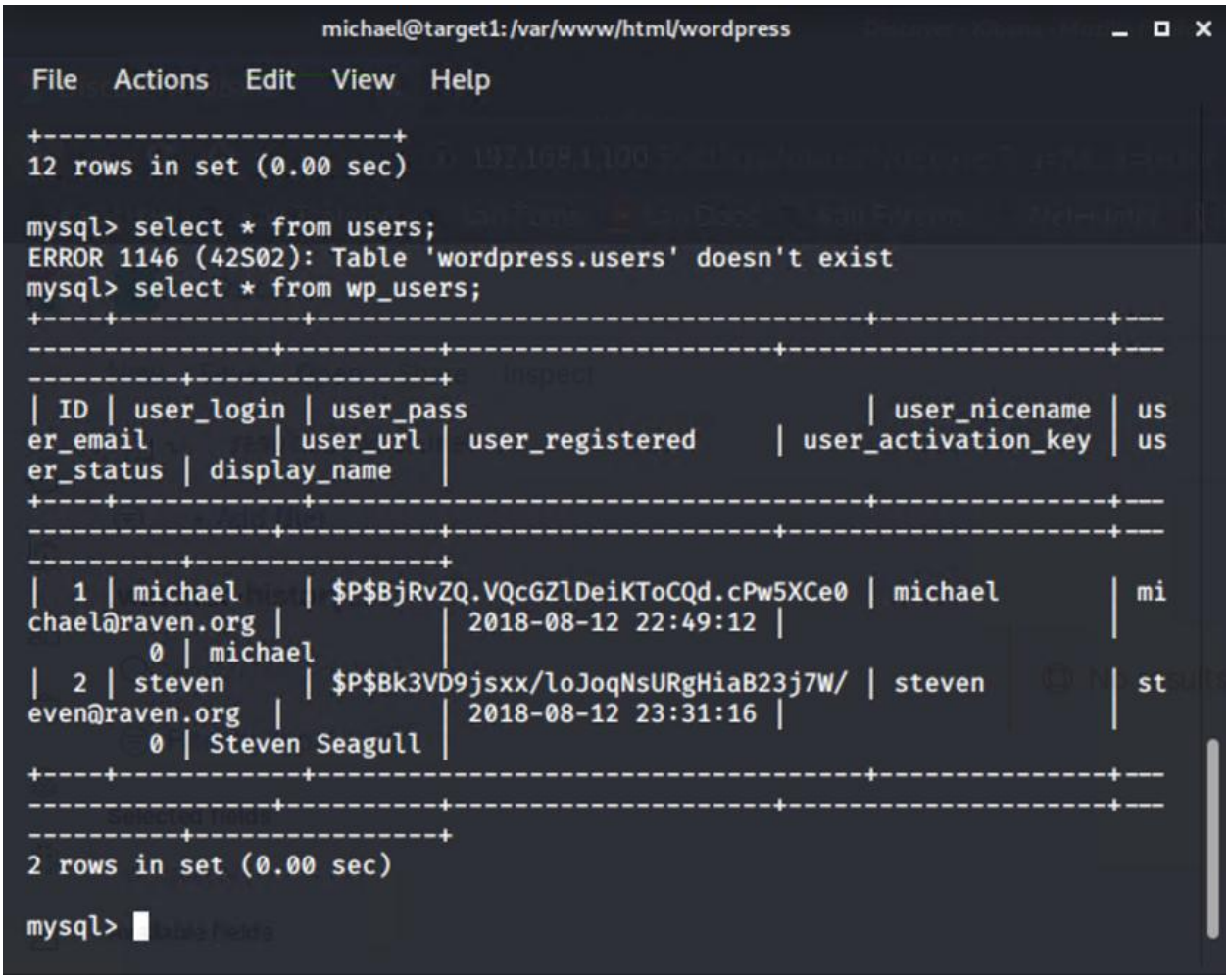
Step 10 - At this point, I open up the *wp_posts* table and find the remaining two flags.

Command line: select * from wp_posts;

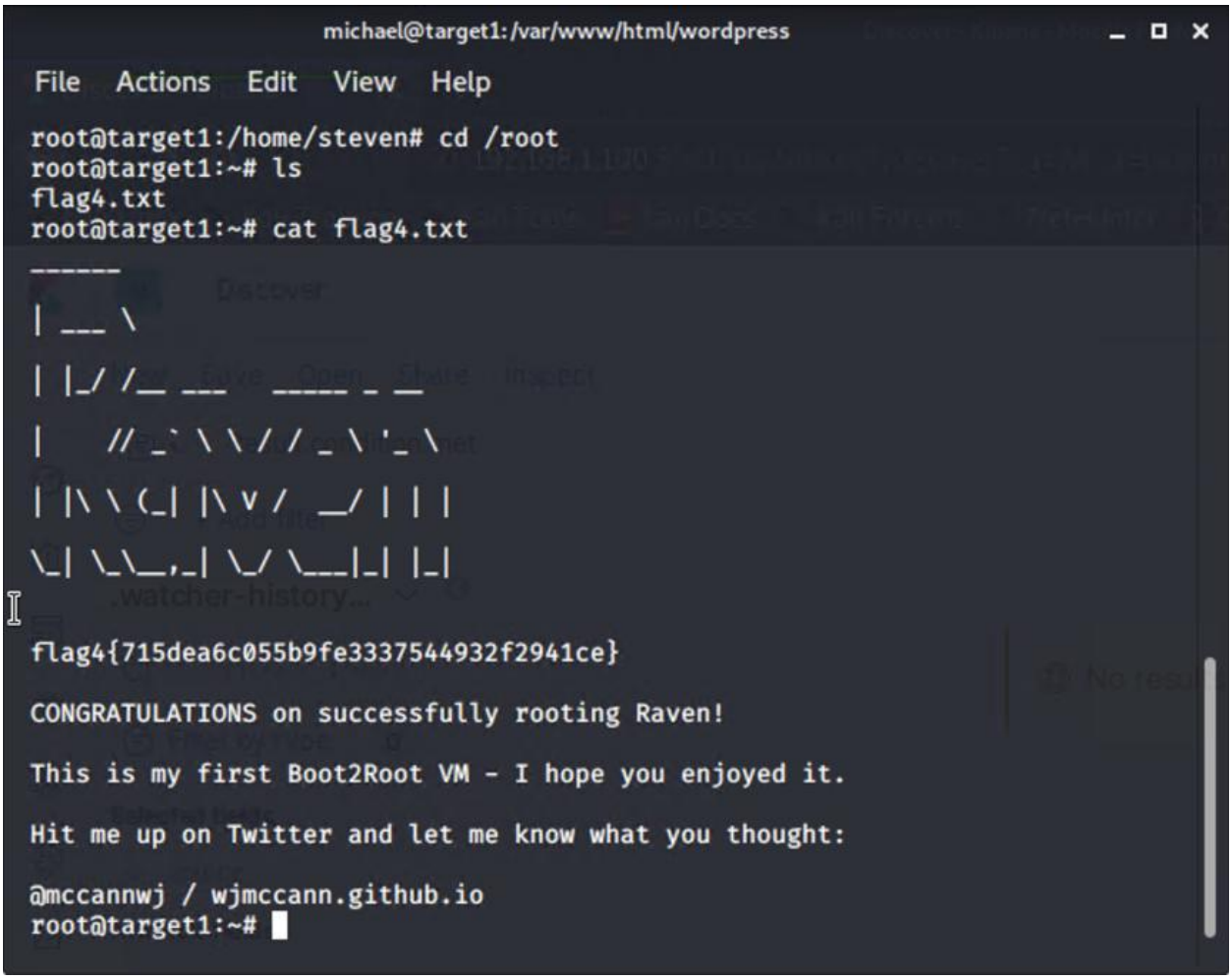


Step 11 - Now that we have the remaining flags, I proceed to investigate the *wp_users* table to obtain the password hashes of both Michael and Steven.

Command line: select * from wp_users;



Step 12 - The final step is to run these password hashes through the password cracking program called John the Ripper to obtain the clear-text passwords and SSH into the system again as Steven, thereby fully rooting the Target 1 machine.



Executive Summary

Overall, the level of security on this system was very poor, and there were likely several other means of exploiting the system which would have gone beyond the scope of this exercise.

First and foremost, the password convention was extremely weak and, for any type of serious enterprise environment, unacceptable. Michael's password being his own username was one of the initial breaking and entering points in the operation, and Steven's was easily cracked with a basic password-cracking application combined with a well-known word list.

Secondly, the lack of updated software exposes the owners of the website to vulnerabilities which have long since been discovered and undoubtedly leveraged against thousands of similar web servers.

Finally, had this been more than just an exercise to display vulnerability, it's almost unquestionable that it would have easily been possible to pivot to another portion of the network and compromise the integrity there, as well. The lack of foresight shown on this machine leads to the immediate conjecture that other machines adjacent to it on the network would be similarly poor in their security posture. As such, this kind of weakness is often symptomatic of a much larger problem which needs to be addressed, if the owners in question are interested in maintaining the integrity of their systems and servers.