SSH Attack and Defense

Custom Project Experiment

Peyton Robinson

10/11/2025

Table of contents

Summary

Given that SSH remains a critical and popular attack vector, this custom project was executed in a home lab environment across three phases: Initial Attack, Defense Implementation, and Test Attack.

The project used the following tools and technologies:

1. Linux OS target and attack machine
2. Splunk Enterprise
3. Iptables
4. Nmap and Metasploit

The timeline includes supporting screenshots, walking through the initiation of attacks and the subsequent construction of the defenses.

Through this experiment, I gained valuable practical experience on how to leverage Splunk and Iptables to effectively mitigate SSH brute force attacks. I concluded that for future hardening, disabling root access and implementing Multi-Factor Authentication (MFA) would further strengthen the defense posture. For future projects, I plan to incorporate more AI tools and prioritize video tutorials over traditional articles for research.

Overview

The SSH protocol is a popular attack vector, and misconfigurations can lead to malicious connections. Once a connection is established for the attacker, a plethora of other attacks could take place: privilege escalation, lateral movement, and data exfiltration to name a few.

I created this custom attack and defense project to learn about Splunk and firewall defenses against brute force attacks on the SSH protocol.

The relevant components are the following:

1. Linux Host device
2. Kali Linux attack virtual machine
3. Splunk Enterprise
4. Iptables
5. Nmap
6. Metasploit

The attack is only to attempt to establish a SSH connection into the host computer on port 22. The attack does not go any further, whether getting into the host computer or not. Defense is only to rate-limit SSH logins from the attacker IP and set up alerts in Splunk.

Attack and Defense Timeline

Timeline Overview

The timeline below is a summary of the project’s steps.

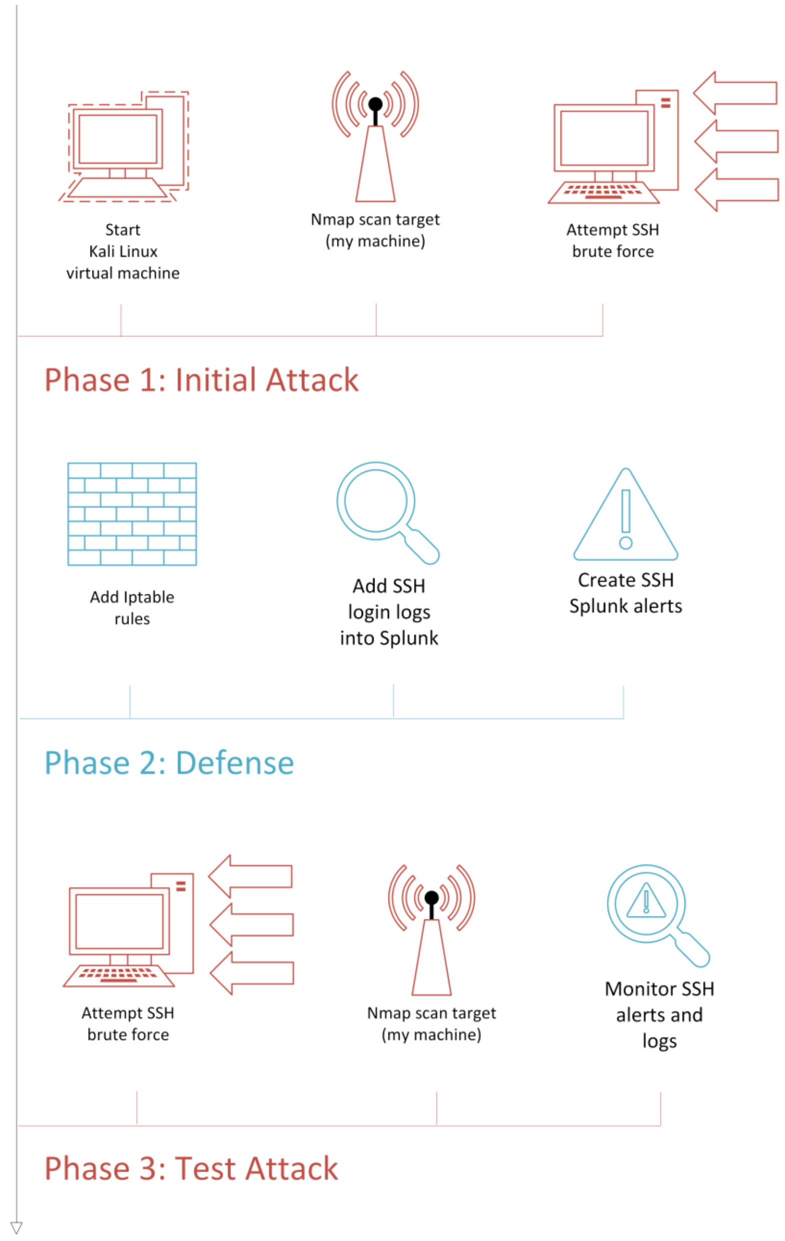


Figure 1 - Microsoft Visio timeline overview of events and phases of attack and defense

Phase 1: Initial Attack

Step 1: Deploy Kali Linux virtual attack machine

To initiate the attack, I started my Kali Linux virtual machine on VirtualBox. This machine was already configured and updated from previous home lab experiments.

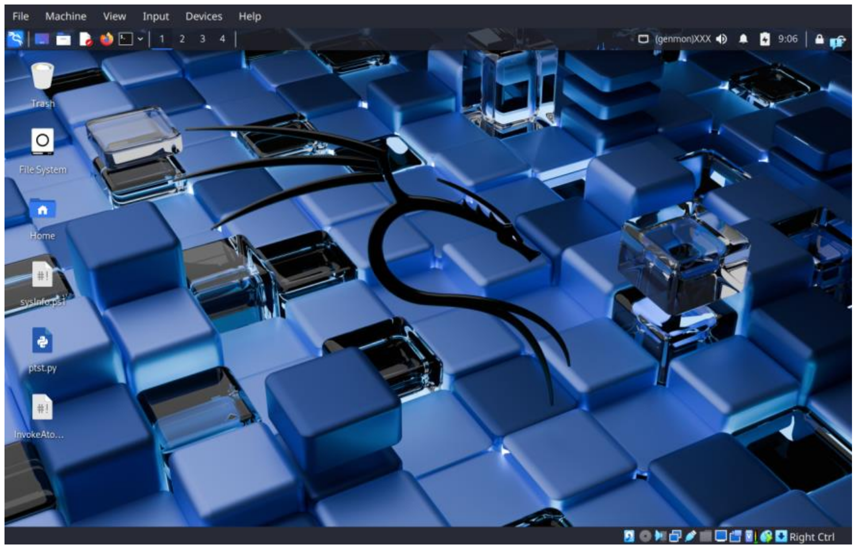
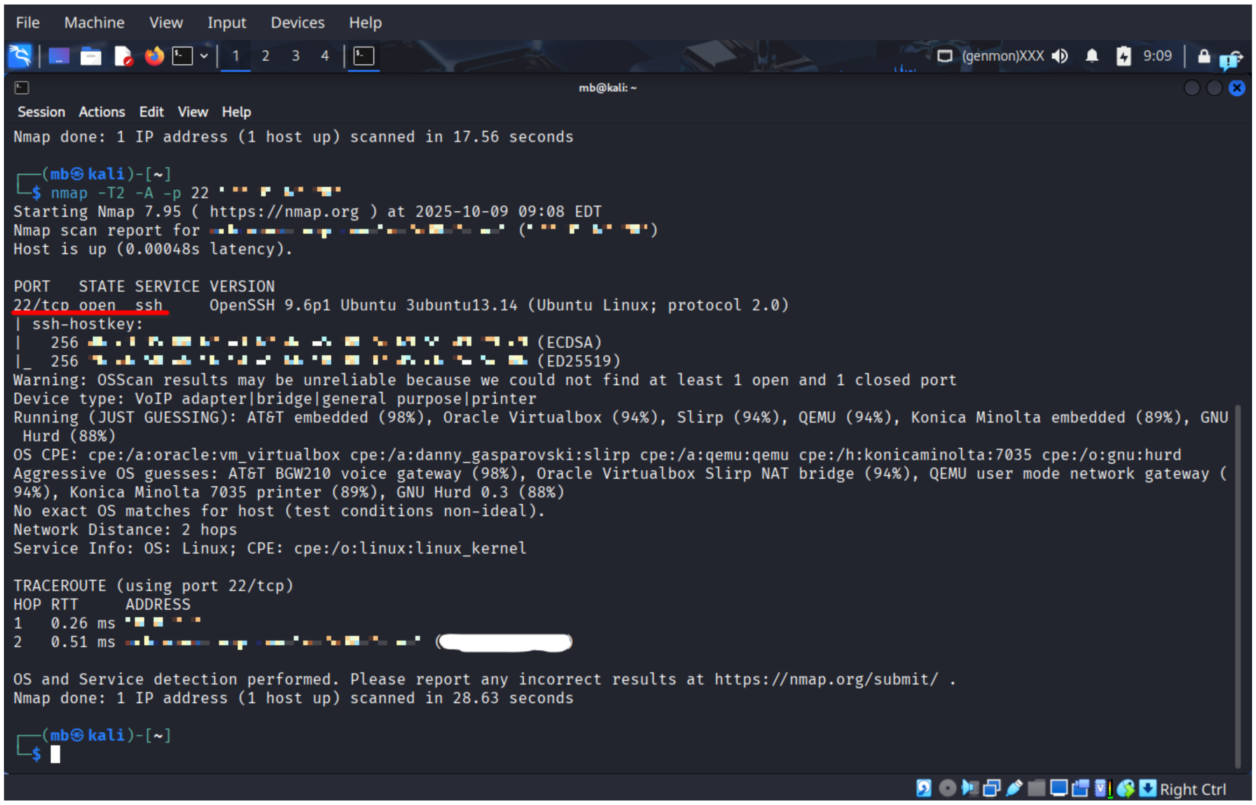


Figure 2 – My Kali Linux home screen.

Step 2: Check port 22 availability with Nmap

Before using Nmap, I installed and enabled SSH on the target machine, which is my personal computer. I used Nmap in the terminal to gather information about port 22 on the target. I also requested extensive machine details by passing the -A flag, and Nmap identified the host as a virtual machine. This misidentification occurred because I bridged the network adapter to the virtual machine, causing Nmap to effectively scan the host's interface.

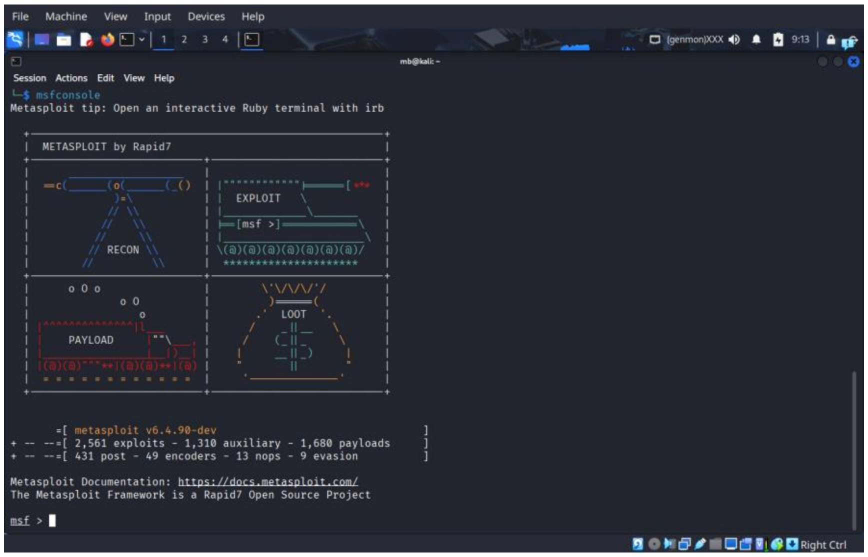

File Machine View Input Devices Help
1 2 3 4
(genmon)XXX 9:09
mb@kali: ~
Session Actions Edit View Help
Right Ctrl

```
Nmap done: 1 IP address (1 host up) scanned in 17.56 seconds

┌──(mb㉿kali)-[~]
└─$ nmap -T2 -A -p 22 [redacted]
Starting Nmap 7.95 ( https://nmap.org ) at 2025-10-09 09:08 EDT
Nmap scan report for [redacted] ([redacted])
Host is up (0.00048s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 9.6p1 Ubuntu 3ubuntu13.14 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 [redacted] (ECDSA)
|_  256 [redacted] (ED25519)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: VoIP adapter|bridge|general purpose|printer
Running (JUST GUESSING): AT&T embedded (98%), Oracle Virtualbox (94%), Slirp (94%), QEMU (94%), Konica Minolta embedded (89%), GNU
 Hurd (88%)
OS CPE: cpe:/a:oracle:vm_virtualbox cpe:/a:danny_gasparovski:slirp cpe:/a:qemu:qemu cpe:/h:konicaminolta:7035 cpe:/o:gnu:hurd
Aggressive OS guesses: AT&T BGW210 voice gateway (98%), Oracle Virtualbox Slirp NAT bridge (94%), QEMU user mode network gateway (
94%), Konica Minolta 7035 printer (89%), GNU Hurd 0.3 (88%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 2 hops
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE (using port 22/tcp)
HOP RTT     ADDRESS
1   0.26 ms [redacted]
2   0.51 ms [redacted] [redacted]

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 28.63 seconds

┌──(mb㉿kali)-[~]
└─$ 
```

Figure 3 – Nmap scan result of the target machine

Step 3: Brute force port 22 with Metasploit

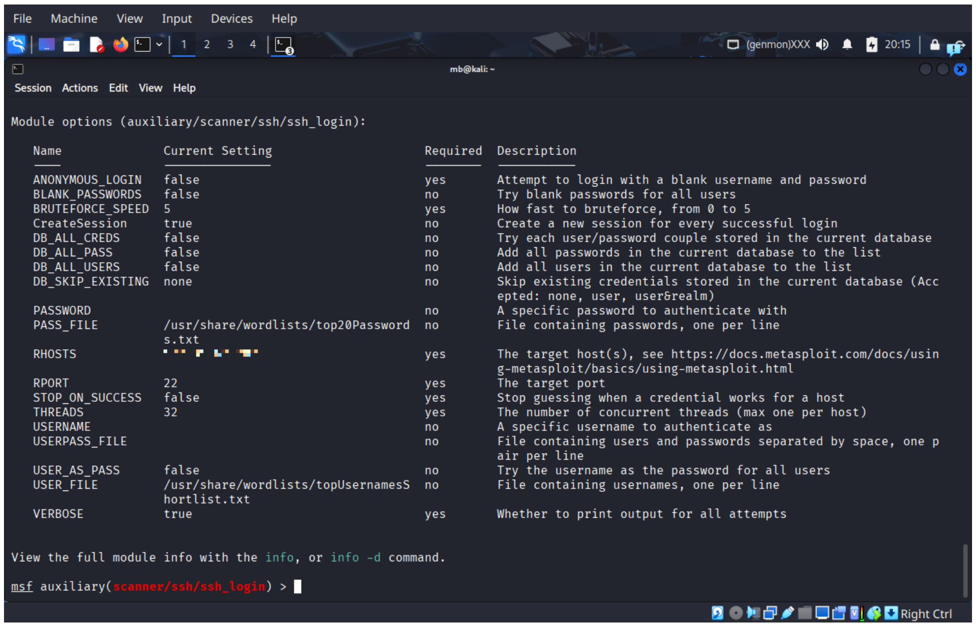
Metasploit is a common penetration testing tool used to automate attacks on devices. I utilized its capability to brute-force SSH connections



```
└─$ msfconsole
Metasploit tip: Open an interactive Ruby terminal with irb

       =[ metasploit v6.4.90-dev                          ]
+ -- --=[ 2,561 exploits - 1,310 auxiliary - 1,680 payloads     ]
+ -- --=[ 431 post - 49 encoders - 13 nops - 9 evasion         ]

Metasploit Documentation: https://docs.metasploit.com/
The Metasploit Framework is a Rapid7 Open Source Project

msf > 
```

Figure 4 – Starting up Metasploit

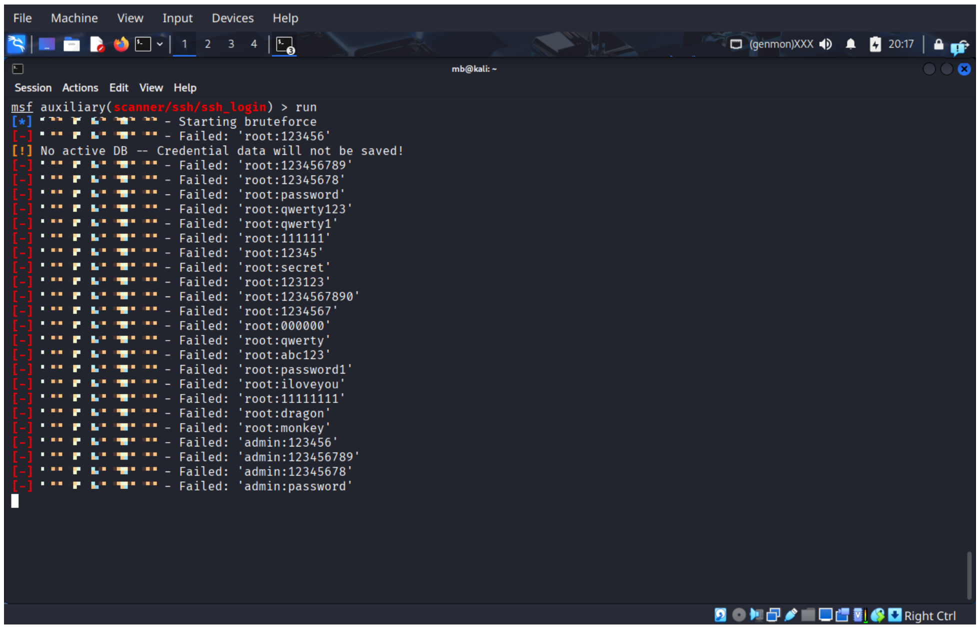
Afterward, I switched to the ssh_login tool and configured the options by setting the host IP, the path to the password list file, the path to the username list file, the number of threads to 32, and the verbosity to true. To expedite the experiment, I created custom passwords and username wordlists, containing only the most popular credentials.



```
Module options (auxiliary/scanner/ssh/ssh_login):

   Name              Current Setting                    Required  Description
   ----              ---------------                    --------  -----------
   ANONYMOUS_LOGIN   false                              yes       Attempt to login with a blank username and password
   BLANK_PASSWORDS   false                              no        Try blank passwords for all users
   BRUTEFORCE_SPEED  5                                  yes       How fast to bruteforce, from 0 to 5
   CreateSession     true                               no        Create a new session for every successful login
   DB_ALL_CREDS      false                              no        Try each user/password couple stored in the current database
   DB_ALL_PASS       false                              no        Add all passwords in the current database to the list
   DB_ALL_USERS      false                              no        Add all users in the current database to the list
   DB_SKIP_EXISTING  none                               no        Skip existing credentials stored in the current database (Acc
                                                                  epted: none, user, user&realm)
   PASSWORD                                             no        A specific password to authenticate with
   PASS_FILE         /usr/share/wordlists/top20Password no        File containing passwords, one per line
                     s.txt
   RHOSTS            [redacted]                         yes       The target host(s), see https://docs.metasploit.com/docs/usin
                                                                  g-metasploit/basics/using-metasploit.html
   RPORT             22                                 yes       The target port
   STOP_ON_SUCCESS   false                              yes       Stop guessing when a credential works for a host
   THREADS           32                                 yes       The number of concurrent threads (max one per host)
   USERNAME                                             no        A specific username to authenticate as
   USERPASS_FILE                                        no        File containing users and passwords separated by space, one p
                                                                  air per line
   USER_AS_PASS      false                              no        Try the username as the password for all users
   USER_FILE         /usr/share/wordlists/topUsernamesS no        File containing usernames, one per line
                     hortlist.txt
   VERBOSE           true                               yes       Whether to print output for all attempts


View the full module info with the info, or info -d command.

msf auxiliary(scanner/ssh/ssh_login) >
```

Figure 5 – Metasploit ssh_login options

I started the attack, and it began passing the passwords and usernames. However, the attack was notably slow, even with a high thread count and sufficient computing power allocated to the VM. After research, I determined this was due to SSH's built-in **brute force mitigations**, which limit login retries. Despite knowing these initial tests would not successfully gain access (as I intentionally withheld the correct credentials), I proceeded with implementing additional security controls to further harden the system.



```
msf auxiliary(scanner/ssh/ssh_login) > run
[*] [redacted] - Starting bruteforce
[-] [redacted] - Failed: 'root:123456'
[!] No active DB -- Credential data will not be saved!
[-] [redacted] - Failed: 'root:123456789'
[-] [redacted] - Failed: 'root:12345678'
[-] [redacted] - Failed: 'root:password'
[-] [redacted] - Failed: 'root:qwerty123'
[-] [redacted] - Failed: 'root:qwerty1'
[-] [redacted] - Failed: 'root:111111'
[-] [redacted] - Failed: 'root:12345'
[-] [redacted] - Failed: 'root:secret'
[-] [redacted] - Failed: 'root:123123'
[-] [redacted] - Failed: 'root:1234567890'
[-] [redacted] - Failed: 'root:1234567'
[-] [redacted] - Failed: 'root:000000'
[-] [redacted] - Failed: 'root:qwerty'
[-] [redacted] - Failed: 'root:abc123'
[-] [redacted] - Failed: 'root:password1'
[-] [redacted] - Failed: 'root:iloveyou'
[-] [redacted] - Failed: 'root:11111111'
[-] [redacted] - Failed: 'root:dragon'
[-] [redacted] - Failed: 'root:monkey'
[-] [redacted] - Failed: 'admin:123456'
[-] [redacted] - Failed: 'admin:123456789'
[-] [redacted] - Failed: 'admin:12345678'
[-] [redacted] - Failed: 'admin:password'
```

Figure 6 – running the brute force attack on the Kali Linux VM

Now that the initial test attack had finished, it was time to harden the system using Iptables and Splunk.

Phase 2: Defense

Step 1: setup Iptable rules

Switching to the host machine, I opened the terminal and added the following iptables rules:

```
sudo iptables -I INPUT -p tcp --dport 22 -m state --state NEW -m
recent --set

sudo iptables -I INPUT -p tcp --dport 22 -m recent -update -
seconds 60 --hitcount 4 -j DROP
```

I used Gemini 2.5 Flash to learn the Iptables syntax and the meaning of each command component. I utilized the -I flag to ensure the rules were inserted at the top of the rule list, which prioritizes them based on how firewall rules are processed. Below is a snapshot of my terminal entering these Iptables rules.

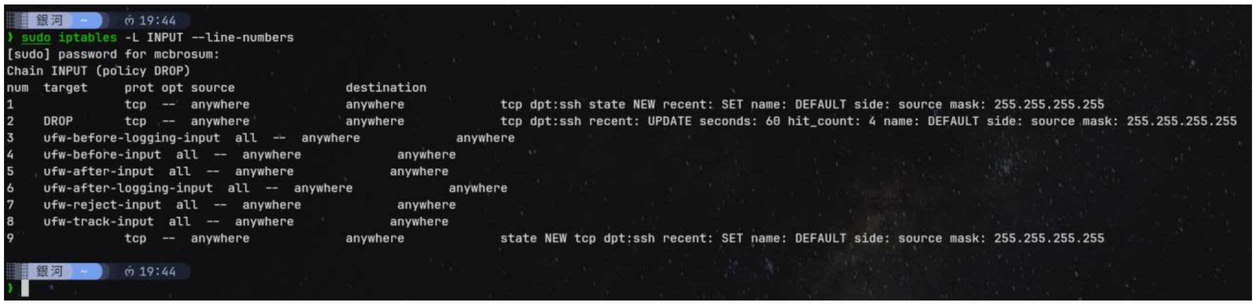
```
[illegible]
```

Figure 7 – adding Iptable rules

To check if the rules were added, I used the following command:

```
sudo iptables -L INPUT --line-numbers
```

The output is below.



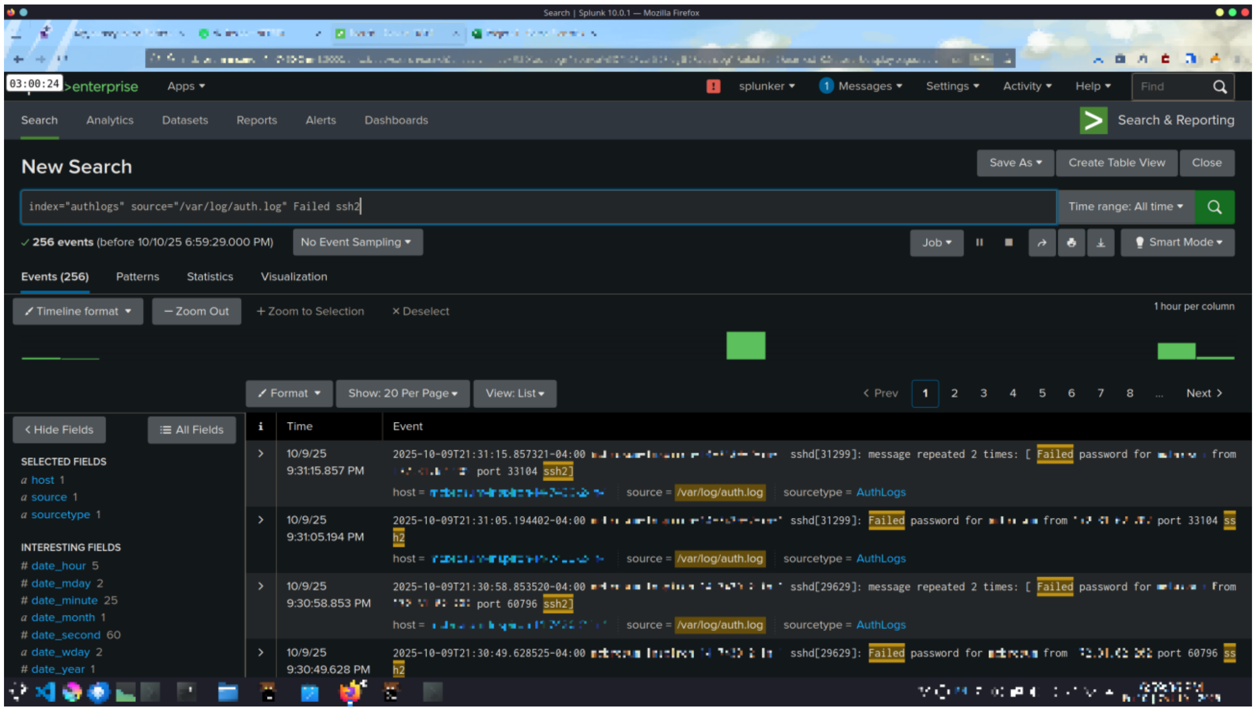
```
 銀河  ~      19:44
❯ sudo iptables -L INPUT --line-numbers
[sudo] password for mcbrosum:
Chain INPUT (policy DROP)
num  target     prot opt source               destination
1               tcp  --  anywhere             anywhere             tcp dpt:ssh state NEW recent: SET name: DEFAULT side: source mask: 255.255.255.255
2    DROP       tcp  --  anywhere             anywhere             tcp dpt:ssh recent: UPDATE seconds: 60 hit_count: 4 name: DEFAULT side: source mask: 255.255.255.255
3    ufw-before-logging-input  all  --  anywhere             anywhere
4    ufw-before-input  all  --  anywhere             anywhere
5    ufw-after-input  all  --  anywhere             anywhere
6    ufw-after-logging-input  all  --  anywhere             anywhere
7    ufw-reject-input  all  --  anywhere             anywhere
8    ufw-track-input  all  --  anywhere             anywhere
9               tcp  --  anywhere             anywhere             state NEW tcp dpt:ssh recent: SET name: DEFAULT side: source mask: 255.255.255.255

 銀河  ~      19:44
❯
```

Figure 8 – Iptable output in the terminal

Next is Splunk monitoring.

Step 2: Add authentication logs into Splunk

I had prior experience with Splunk, so I was familiar with the basics of the UI; however, I needed to study the search syntax more deeply. Gemini 2.5 Flash was helpful for learning how to search logs, and I also followed a specific Splunk guide for searching SSH logs [1]. Below is my first search for invalid SSH login attempts:



```
index="authlogs" source="/var/log/auth.log" Failed ssh2
```

✓ 256 events (before 10/10/25 6:59:29.000 PM)

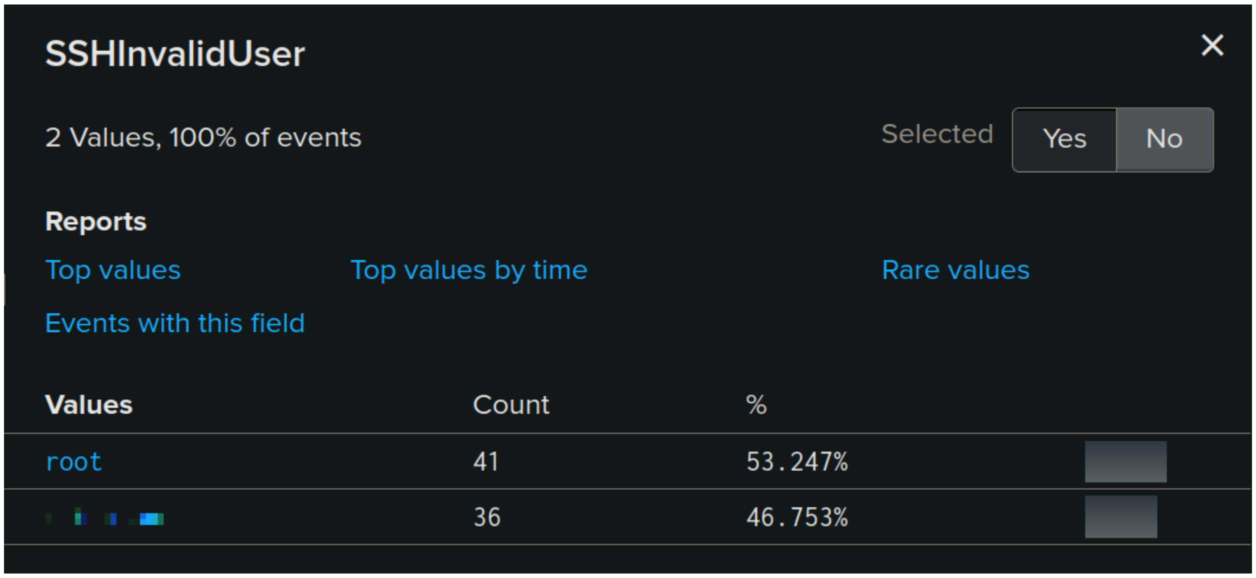
SELECTED FIELDS
a host 1
a source 1
a sourcetype 1

INTERESTING FIELDS
# date_hour 5
# date_mday 2
# date_minute 25
a date_month 1
# date_second 60
a date_wday 2
# date_year 1

```
10/9/25      2025-10-09T21:31:15.857321-04:00 [redacted] sshd[31299]: message repeated 2 times: [ Failed password for [redacted] from
9:31:15.857 PM  [redacted] port 33104 ssh2]
             host = [redacted]  source = /var/log/auth.log  sourcetype = AuthLogs
10/9/25      2025-10-09T21:31:05.194402-04:00 [redacted] sshd[31299]: Failed password for [redacted] from [redacted] port 33104 ss
9:31:05.194 PM  h2
             host = [redacted]  source = /var/log/auth.log  sourcetype = AuthLogs
10/9/25      2025-10-09T21:30:58.853520-04:00 [redacted] sshd[29629]: message repeated 2 times: [ Failed password for [redacted] from
9:30:58.853 PM  [redacted] port 60796 ssh2]
             host = [redacted]  source = /var/log/auth.log  sourcetype = AuthLogs
10/9/25      2025-10-09T21:30:49.628525-04:00 [redacted] sshd[29629]: Failed password for [redacted] from [redacted] port 60796 ss
9:30:49.628 PM  h2
```

Figure 9 – Splunk search results with "Failed" and "ssh2" keywords.

After finding the correct logs, I extracted a field and named it `SSHInvalidUser` by using regex. Adding this field to the search result creates this chart:



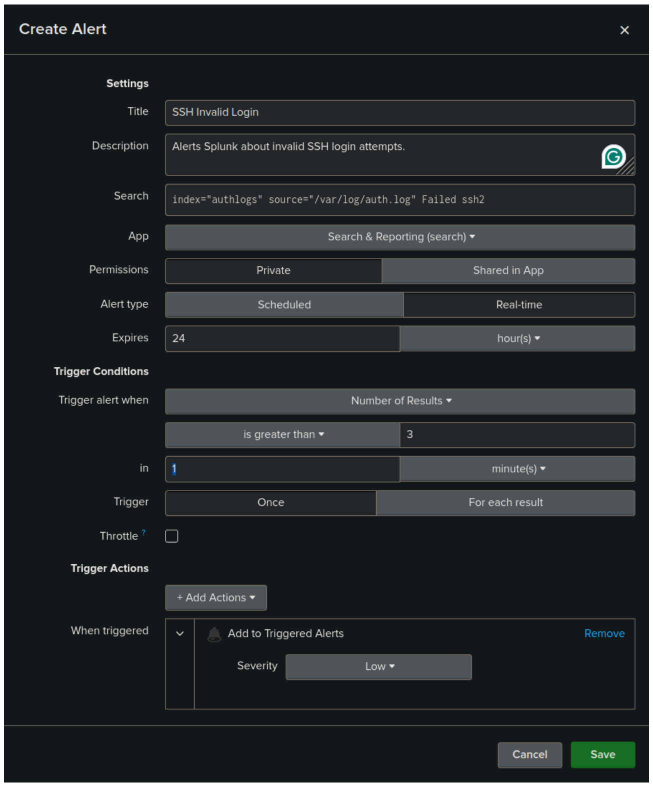
2 Values, 100% of events

Values	Count	%
root	41	53.247%
[redacted]	36	46.753%

Figure 10 – chart showing invalid users from the SSH attack.

Step 3: Create Splunk alerts for SSH login attempts

Once I found the correct logs, I created real-time alerts using the SSH log search query. Below is a depiction of me setting these alerts up:



Create Alert

Settings

Title: SSH Invalid Login

Description: Alerts Splunk about invalid SSH login attempts.

Search:
```
index="authlogs" source="/var/log/auth.log" Failed ssh2
```

App: Search & Reporting (search)

Permissions: [✓ Private] Shared in App

Alert type: Scheduled [✓ Real-time]

Expires: 24 hour(s)

Trigger Conditions

Trigger alert when: Number of Results

is greater than 3

in 1 minute(s)

Trigger: [✓ Once] For each result

Throttle: ☐

Trigger Actions

+ Add Actions

When triggered: Add to Triggered Alerts — Remove

Severity: Low

Cancel Save

Figure 10 – setting up real-time SSH Splunk alerts.

Now that my defenses are set up, I can initiate a test attack.

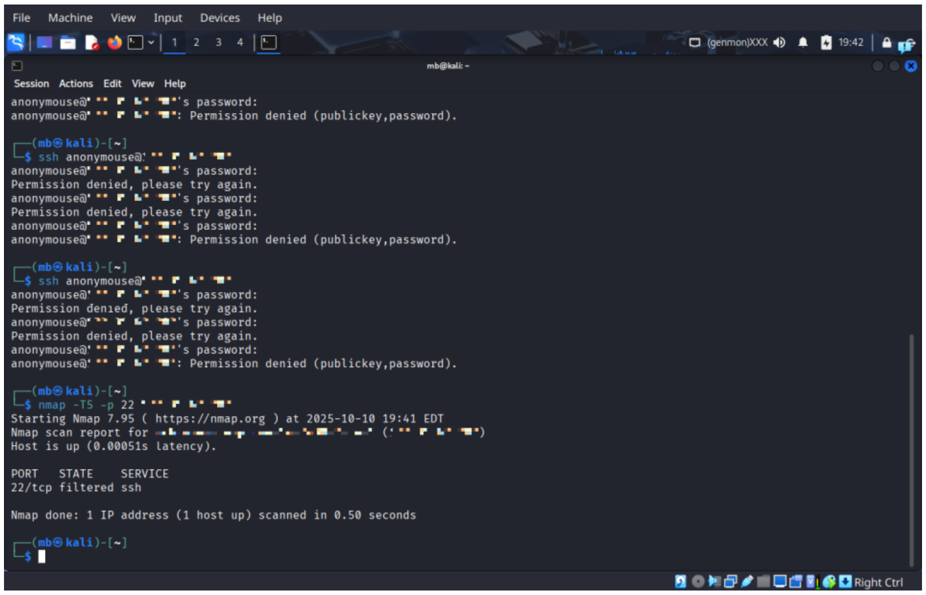
Phase 3: Test Attack

Step 1: Brute force port 22

Using the same Kali Linux virtual machine and method as before, I attempted the same attack on the target machine. This time, I manually inputted random passwords instead of using Metasploit. This was done specifically to trigger the Splunk alerts and verify that the firewall dropped my packets as intended.

Step 2: View port 22 with Nmap after attack

Going back to the attack machine, I ran another Nmap scan after the attack. The port state is now reported as "filtered," meaning that packets from the attack machine are being dropped before reaching the target machine. The attack machine will now have to wait one minute before it can make four more attempts.

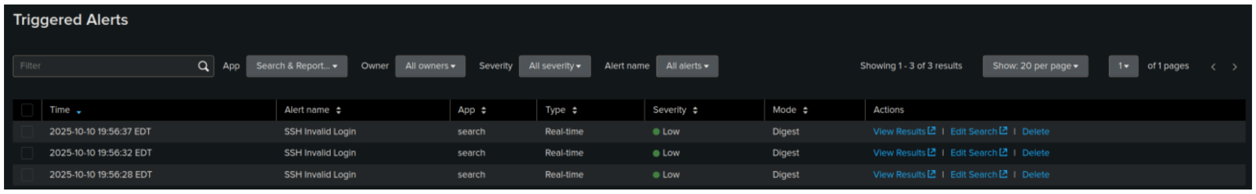


```
anonymouse@[redacted]'s password:
anonymouse@[redacted]: Permission denied (publickey,password).

┌──(mb㉿kali)-[~]
└─$ ssh anonymouse@[redacted]
anonymouse@[redacted]'s password:
Permission denied, please try again.
anonymouse@[redacted]'s password:
Permission denied, please try again.
anonymouse@[redacted]'s password:
anonymouse@[redacted]: Permission denied (publickey,password).

┌──(mb㉿kali)-[~]
└─$ ssh anonymouse@[redacted]
anonymouse@[redacted]'s password:
Permission denied, please try again.
anonymouse@[redacted]'s password:
Permission denied, please try again.
anonymouse@[redacted]'s password:
anonymouse@[redacted]: Permission denied (publickey,password).

┌──(mb㉿kali)-[~]
└─$ nmap -T5 -p 22 [redacted]
Starting Nmap 7.95 ( https://nmap.org ) at 2025-10-10 19:41 EDT
Nmap scan report for [redacted] ([redacted])
Host is up (0.00051s latency).

PORT   STATE    SERVICE
22/tcp filtered ssh

Nmap done: 1 IP address (1 host up) scanned in 0.50 seconds

┌──(mb㉿kali)-[~]
└─$ 
```

Figure 12 – An Nmap scan showing a filtered state on port 22.

Step 2: Monitor SSH login alerts in Splunk

While the attack was live, I observed alerts on Splunk. Splunk successfully generated alerts based on the failed SSH logins.



Triggered Alerts

App: Search & Report... Owner: All owners Severity: All severity Alert name: All alerts

Showing 1 - 3 of 3 results Show: 20 per page 1 of 1 pages

Time	Alert name	App	Type	Severity	Mode	Actions
2025-10-10 19:56:37 EDT	SSH Invalid Login	search	Real-time	● Low	Digest	View Results | Edit Search | Delete
2025-10-10 19:56:32 EDT	SSH Invalid Login	search	Real-time	● Low	Digest	View Results | Edit Search | Delete
2025-10-10 19:56:28 EDT	SSH Invalid Login	search	Real-time	● Low	Digest	View Results | Edit Search | Delete

Figure 11 – A view of custom Splunk alerts.

This concludes the experiment.

Outcome

As my first project, testing and hardening my machine was an enjoyable and valuable exercise. I gained significant knowledge and look forward to performing more such experiments in the future.

Lessons Learned

The following is a summary of what I learned in this project:

1. Splunk
    a. How to add data specify which files to monitor
    b. How to search for logs using regex, specifying the source
    c. How to extract fields from logs
    d. How to create and view alerts
2. SSH
    a. SSH's built in brute force protection
    b. How to download and start SSH
3. Virtual Machine
    a. How the virtual machine gets Wi-Fi connection implicitly through the host
4. Iptables
    a. Overall syntax
    b. Listing rules
    c. Creating new rules

I used online resources and Gemini Pro to learn and practice for this project.

Recommendations

For further hardening of SSH connections, you can disable root login and require Multi-Factor Authentication (MFA). Disabling root login prevents attackers from immediately gaining a privileged shell, and MFA provides additional mitigation against brute-force attacks.

For future projects, I will increase my use of AI for learning tool usage and rely more on online video tutorials for implementing security controls. Reading numerous articles that did not directly apply to the specific situation extended the overall project work time.

References

[1]

D. Miessler, “Monitoring SSH Bruteforce Attempts Using Splunk,” *Danielmiessler.com*, 2017. https://danielmiessler.com/blog/monitoring-ssh-bruteforce-attempts-using-splunk (accessed Oct. 11, 2025).