

Live Detection and Mitigation for SSH Brute Force Attacks

Nael Zahreldine¹, Amer Jaber¹, Ahmad Hani El Fawal^{1,2}, Dani Merhi Bleik¹, Ali Mansour²
nael.zahreldine@mubs.edu.lb, amer.jaber@mubs.edu.lb, elfawal@ieee.org, dani.merhibleik@mubs.edu.lb, mansour@ieee.org

¹ School of Computer and Applied Sciences, Modern University for Business and Science (MUBS), Damour 5660, Lebanon.

² LABSTICC, UMR 6285 CNRS, ENSTA-IP Paris, Brest, France.

Abstract—Secure Shell (SSH) is a critical remote management protocol, but its pervasiveness makes it a prime target for brute-force, scan, and exploitation attacks. Conventional defenses such as static firewall rules or post-malicious-event log analysis are often reactive and can be easily bypassed. The aim of this paper is to provide an efficient, multi-layered defense system for securing SSH services on Ubuntu servers based on both proactive and reactive mechanisms. The proposed architecture combines NFtables-based hardening of the base image, Port Knocking to obscure the SSH port from scanners without legitimate access, and automated countermeasures using Fail2Ban and IPTables rate limiting. At its very core, a lightweight Python detection module employs PyShark to perform live packet inspection, facilitating early pattern detection in seconds. The system was validated in a controlled testbed setup using Kali Linux as the attacking host. Simulations indicate that an unprotected server left alone was compromised within a second, while the proposed framework successfully detected and countered attacks in real-time. The solution offers an adaptive, adaptive, and resource-efficient approach to improving SSH security against emerging threats.

Keywords— *SSH Security, Intrusion Detection, Brute-Force Attacks, Port Knocking, Live Packet Analysis, Defense-in-Depth*

I. INTRODUCTION

Secure Shell (SSH) remains a fundamental protocol for secure remote access to system administrators. Operating on Transmission Control Protocol (TCP) port 22 as a default [1][2], SSH employs secure cryptographic methods for keeping communications confidential and secure in terms of integrity. Employed extensively for manual server installation to deployment scripts run automatically, its importance renders it a first choice target too. Across the internet, automated bots, brute-force password crackers, and network scanners continuously probe SSH services for vulnerabilities [3].

One of the main disadvantages of conventional defense mechanisms is that they rely on single-point solutions. Fail2Ban, perhaps the most widely installed countermeasures, operates reactively by maintaining authentication logs and blocking Internet Protocol (IP) addresses following a number of failed login attempts [4]. While effective in thwarting high-speed brute-force attacks, it can be skirted by low-speed distributed attacks below detection levels. Similarly, a basic firewall may prevent unwanted traffic but is of a little use in defending against such attacks from trusted IP addresses or those exploiting the SSH service itself. These restrictions reaffirm the necessity for a more responsive, tiered defense system based on multi mechanisms.

The approach proposed here is not a universal solution, but rather a defense-in-depth solution for Ubuntu servers that will inculcate proven and tested security tools in an intelligent manner. Port Knocking is used to make the SSH port invisible to unauthorized scanners by making it obscure. A custom Python module, utilizing PyShark which has a direct interface with Wireshark that captures and analyzes live network packets. Based on a predefined time-based threshold within the code, the module can flag traffic patterns as anomalous, thereby enabling the detection of potentially malicious behavior before such patterns are obvious within system logs. For intrusive attempts to bypass prior layers in a host-based manner, Fail2Ban inadvertently counteracts the threat by integrating kernel-level rate limiting. The resulting system offers a practical, pragmatic, and resource-conscious solution that enhances resilience without the sophistication of operations or computational expense typically associated with large-scale machine learning systems.

II. RELATED WORK

Protection of SSH against brute-force attacks is an extensively researched problem with multiple strategies proposed through the years. The standard defenses against SSH brute-force attacks is based on reactive host-based tools. Well-known solutions such as Fail2Ban which is an intrusion prevention tool that scans log files for suspicious activity and automatically updates firewall rules to block offending IPs [7]. When such tools sense an excessively large number of login failures from a single IP address, they insert firewall rules dynamically that block the attacking IP. While this is a simple and effective solution against noisy high-speed attacks, it has a drawback — the attacker must first initiate and succeed partially in the attack before such defenses kick in [5]. Consequently, slow or distributed brute-force attacks can remain undetected, hiding low until massive damage has already been caused. Another method is to analyze network traffic for signs of malicious behavior. Honeypots such as Cowrie [5][6] are dummy systems intended to attract attackers in a way that allows security researchers to examine the attack techniques without risking real servers. However, although honeypots (fake systems that lure attackers to study their behavior safely) are beneficial in the gathering of intelligence, they do not defend production environments per se. Meanwhile, with developments in machine learning (ML), it became possible to generate sophisticated Intrusion Detection Systems (IDS) that have the ability to identify unapproved SSH activity accurately [7]. Though showing encouraging performance, ML-based systems have significant drawbacks: they require massive clean training sets, plenty of computational capacity, and complex tuning, limiting their real-world application, especially in resource-constrained environments [8].

Proactive defenses try to prevent attacks before they even begin. Port Knocking is a classic example of a proactive method [9]. With it, the SSH port is never exposed and is protected behind the firewall. Trusted users must send a sequence of connection requests ("knocks") to a specified sequence of closed ports. If the sequence is successful, the firewall temporarily opens the SSH port for the user. The method provides a form of security through obscurity that effectively prevents automatic scanners from discovering the SSH service on its default port 22 [10]. However, while Port Knocking significantly reduces attack surface exposure, it is not foolproof and may introduce usability challenges. Modern adaptations have extended this concept to environments such as mobile cloud computing, enhancing flexibility and robustness [11].

Each of these solutions addresses SSH security from a different perspective, but none offers a complete defense on its own. Host-based reactive systems are slow and vulnerable to stealthy attacks, ML-based IDS at the network level are complicated and resource-intensive, and proactive obfuscation techniques like Port Knocking rely on confidentiality and can degrade usability. Our work addresses these loopholes by integrating the strengths of these solutions into one unified, hybrid defense system. Specifically, our solution utilizes proactive port obscurity through Port Knocking, in-real-time network-level protection through a lightweight Python script that avoids the expense of ML, and efficient reactive countermeasures through Fail2Ban and kernel-level rate limiting. This multi-layered defense-in-depth framework provides stronger, more robust server-side security than single-method or edge-based alternatives [12], offering practical, scalable security suitable for real-world deployment.

III. METHODOLOGY

In an effort to hide the SSH service from scanners, we employed Port Knocking using the knockd service. This technique leaves the SSH port closed and hidden behind the firewall until a known user initiates a secret series of connection attempts to specific shut-down ports, say ports 11001, 11002, followed by port 11003 (using a modified secret sequence of knocks to communicate with the firewall, the port opens or closes depending on the correctness of the knock sequence). The knockd merely executes a script which opens port 22 for a temporary period only for the IP address when knockd notices this same sequence from an IP address [13]. A subsequent knock sequence will shut down the port again. This method successfully conceals the SSH port from typical network scanning tools such as Network Mapper (nmap), greatly reducing vulnerability to automated attacks by keeping unwanted connection attempts away from the SSH daemon.

The central innovation in our system is the real-time detection and throttling layer, which sees network traffic directly in order to be able to react instantly to suspicious activity. We have developed a custom Python script using PyShark to perform live packet sniffing those targets attempts of new SSH connections, i.e., TCP SYN packets directed towards port 22. The script has a gradual setup: it determines whether there are five login attempts from the same IP in 10 seconds—specified in the Python script—and then triggers

alerts through a terminal notification and an email to the system administrator. The threshold works because it is unlikely for a human to make five attempts within a short space of time. This early detection provides for instant action before the attacker can amplify the attack.

To complement this early detection, we added a second aggressive throttling mechanism using iptables' recent module to track repeated connection attempts (enforces blocks by dropping the packets). Fail2Ban monitors these events and dynamically blocks IPs that make over five attempts in sixty seconds, helping to prevent brute-force attacks. The alert level for this rate limiting is maintained higher than the alert level of the script to ensure that the custom detection script always provides the initial warning, with iptables as the follow-up block that stops committed attackers.

With these proactive and in-real-time measures in place, persistent attackers are met with a reactive, application-level defense through Fail2Ban. Configured to scan for the SSH authentication logs in /var/log/auth.log, Fail2Ban is scanning for several failed login attempts. Once it hits five attempts from the same IP address, Fail2Ban bans the IP for one hour, prohibiting login attempts after that. This is the final defense in case initial detection and slowdown are circumvented, repeated malicious endeavor is definitely blocked.

Combined together, these layers form a hybrid, defense-in-depth architecture that comprises proactive port obfuscation, live packet analysis in real-time, and reactive application-level blocking. Our innovation lies in combining these proven methods into an integrated, automated system with the power of a lightweight custom detection script. This approach fills the gaps of existing solutions—offering faster detection than traditional log-based solutions, going around the complexity of machine learning systems, and extending basic port hiding to encompass responsive and adaptive strategies. The result is an pragmatic, scalable, and robust SSH security model poised for real-world deployment.

The overall sequence of Port Knocking, PyShark-based detection, and Fail2Ban-initiated response mechanisms provides multi-layered protection against brute-force attacks. This layered approach maintains the security of the SSH port. The overall process of integrating these proactive, real-time, and reactive mechanisms is illustrated in Figure 1.

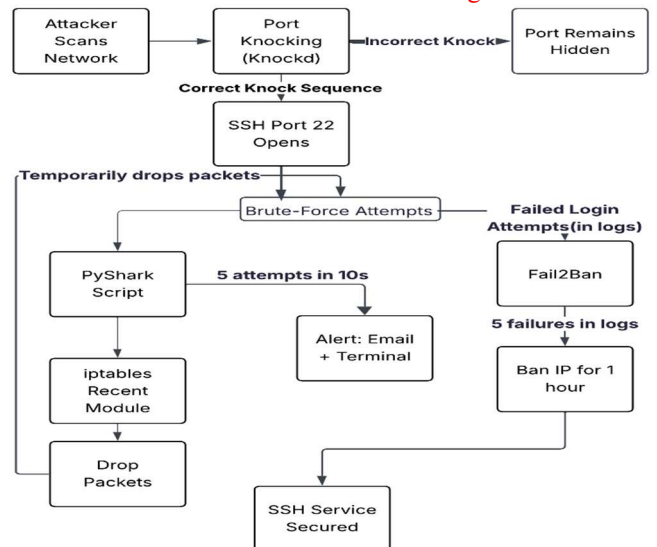


Fig. 1. Workflow of the proposed SSH defense system

IV. SIMULATIONS AND RESULTS

In order to verify the effectiveness of our multi-layer defense, we created a test to simulate a real-world SSH brute-force attack. We aimed to compare system resilience before and after our installation of the multi-layer security setup.

The testing setup consisted of two computers connected in the same virtual network: an Ubuntu Server 24.04.2 LTS-based victim server and a Kali Linux 2023.4-based attacker computer. The attacker computer included standard penetration testing tools: Address Resolution Protocol (arp-scan) for scanning the network, nmap for port scanning, and Hydra for brute-forcing SSH credentials. The setup was utilized to realistically mimic the capabilities of a typical attacker targeting an open SSH service. There were two experiments. In the baseline test, we made an attack on a fresh Ubuntu installation with default settings and no additional security features. In the hardened test, we replicated the same attack actions against a server that had been hardened with our multilayered defense. The baseline test results were not surprising but important as a check. The open server was compromised in a matter of seconds. The scan from the attacker immediately caught sight of the open SSH port, allowing Hydra to execute password guess attacks at full speed. Figure 2 shows the nmap scan report from the baseline test, confirming that port 22 (SSH) was open and exposed.

```
(kali㉿kali)-[~]
$ nmap -p 22 192.168.18.131
Starting Nmap 7.95 ( https://nmap.org ) at 2025-08-12 06:12 EDT
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled.
ervers
Nmap scan report for 192.168.18.131
Host is up (0.0042s latency).

PORT      STATE SERVICE
22/tcp    open  ssh
MAC Address: 00:0C:29:35:DC:64 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.28 seconds
```

Fig. 2. Nmap scan report on the unprotected server

A valid credential match was found, giving the attacker an entire interactive shell with no hindrance. In the hardening case, each stage of the attack was trapped by a different layer of protection. The first probing attempt through a nmap scan. Figure 3 shows that the nmap scan revealed nothing, as the SSH port was completely hidden and filtered due to Port Knocking.

```
(kali㉿kali)-[~]
$ nmap -p 22 192.168.18.131
Starting Nmap 7.95 ( https://nmap.org ) at 2025-08-12 06:36 EDT
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled.
ervers
Nmap scan report for 192.168.18.131
Host is up (0.0012s latency).

PORT      STATE      SERVICE
22/tcp    filtered  ssh
MAC Address: 00:0C:29:35:DC:64 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.67 seconds
```

Fig. 3. Nmap scan report on the protected server

To test the rest of the layers, we bypassed Port Knocking manually and initiated a brute-force attack using Hydra. Within 2–3 seconds, the Python intrusion detection script, running PyShark to capture packets in real-time, identified the suspicious pattern of connection and raised both terminal alert and email notification. Figure 4 displays the terminal output from our real-time detection script, showing multiple SSH brute-force attempts from different source IPs.

```
(myenv) root@user-VMware-Virtual-Platform:/home/user/Desktop# python3 ssh_detection_alert.py  
Monitoring SSH traffic for brute-force attacks...  
 ALERT: SSH Brute Force Attack Detected!   
 Source IP: 192.168.18.128 | Attempts: 5 within 10 seconds  
 Attempting to send email...  
 Email alert sent to [redacted]@gmail.com  
 ALERT: SSH Brute Force Attack Detected!   
 Source IP: 192.168.18.128 | Attempts: 5 within 10 seconds  
 Attempting to send email...  
 Email alert sent to [redacted]@gmail.com  
 ALERT: SSH Brute Force Attack Detected!   
 Source IP: 192.168.18.128 | Attempts: 5 within 10 seconds  
 Attempting to send email...  
 Email alert sent to [redacted]@gmail.com  
 ALERT: SSH Brute Force Attack Detected!   
 Source IP: 192.168.18.128 | Attempts: 5 within 10 seconds  
 Attempting to send email...  
 Email alert sent to [redacted]@gmail.com  
 ALERT: SSH Brute Force Attack Detected!   
 Source IP: 192.168.18.128 | Attempts: 5 within 10 seconds
```

Fig. 4. Terminal output from the real-time detection script

Figure 5 shows the email alert received by the system administrator, triggered by this detected attack, which includes the source IP and the detected behavior pattern.



Fig. 5. An example of the email alert received by the administrator

Once the attacker persisted, IPTables rate-limiting rules stepped in, dropping spurious connection packets and slowing down the brute-force attack significantly. Figure 6 displays the iptables configuration with the recent module, demonstrating how the system was configured to drop packets from repeat offender

```
root@user-VMware-Virtual-Platform:/home/user# iptables -L -v -n
Chain INPUT (policy ACCEPT 165K packets, 562M bytes)
pkts bytes target prot opt in out source destination
165K 562M ufw-before-logging-input 0 -- * * 0.0.0.0/0 0.0.0.0/0
165K 562M ufw-before-input 0 -- * * 0.0.0.0/0 0.0.0.0/0
165K 562M ufw-after-input 0 -- * * 0.0.0.0/0 0.0.0.0/0
165K 562M ufw-after-logging-input 0 -- * * 0.0.0.0/0 0.0.0.0/0
165K 562M ufw-reject-input 0 -- * * 0.0.0.0/0 0.0.0.0/0
165K 562M ufw-track-input 0 -- * * 0.0.0.0/0 0.0.0.0/0
347 28564 6 -- * * 0.0.0.0/0 0.0.0.0/0 tcp dpt:22 stat
T name: DEFAULT side: source mask: 255.255.255.255
326 19496 DROP 6 -- * * 0.0.0.0/0 0.0.0.0/0 tcp dpt:22 stat
DATE seconds: 60 hit_count: 4 name: DEFAULT side: source mask: 255.255.255.255
```

Fig. 6. The IPTables configuration showing the rate-limiting rule

Figure 6 displays the output of the iptables -L -v -n command, which shows the active firewall rules on a Linux system. The most critical part for a security context is the Chain INPUT section, as it governs incoming network traffic. Here, we can see two specific rules targeting TCP port 22 (the default for SSH). The first rule, which is not fully visible, likely accepts new connections. The second rule is a rate-limiting rule that uses the recent module to track connection attempts. It is configured to DROP packets from a source IP if it has initiated more than a certain number of connections to port 22 within a specific time frame, as indicated by the hit_count: 4 and seconds: 60 parameters. This rule is a key defensive measure against SSH brute-force attacks, as it automatically blocks a malicious host after four attempts in one minute.

Slowing down the speed of attacks broke Hydra's efficiency, making it pointless for the attacker to continue. Finally, after five failed login attempts successfully reaching the SSH service, Fail2Ban stepped in and implemented a one-hour ban on the attacking IP addresses shutting down the session. Figure 7 illustrated the Fail2ban operation.

```
root@user-VMware-Virtual-Platform:/home/user# fail2ban-client status sshd
Status for the jail: sshd
|- Filter
| |- Currently failed: 0
| |- Total failed: 16
| `-- Journal matches: _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
  |- Currently banned: 2
  |- Total banned: 3
  `-- Banned IP list: 192.168.18.128 192.168.18.1
root@user-VMware-Virtual-Platform:/home/user#
```

Fig. 7. Fail2ban Banning the IP Addresses

This sequence of defenses demonstrated not just the personal usefulness of each layer, but also the advantage of their coordination. The Port Knocking mechanism denied initial exposure, the real-time detection script provided early warning before it might escalate, and Fail2Ban acted as a mandatory application-level block. The test confirmed that a well-balanced selection of open-source applications, if installed in a unified way, can be effective, real-time protection against SSH brute-force attacks without having to resort to expensive or proprietary security software. The results point towards the value of uniting proactive, reactive, and detection features into creating a robust and realistic defense-in-depth posture. The terminal output captured during the test attack shows that the detection script caught multiple SSH connection attempts within the specified threshold and confirming that an email alert was generated for each instance.

V. DISCUSSION

Our simulation results offer compelling evidence for the efficacy of our multi-layered defense-in-depth solution. The experiments, conducted in an isolated virtual testbed environment, were found to demonstrate a palpable and measurable difference in system robustness between an unprotected server and the one protected with our proposed solution.

The baseline test confirmed the vulnerability of an ordinary Ubuntu server. The server was easily scanned for the open SSH port 22 using a nmap scan. This allowed a brute-force tool to gain access into the server within seconds. This proves the need for a strong security posture. The hardened server, however, managed to block all stages of the attack.

The Port Knocking layer effectively concealed port 22 from initial nmap scans, hiding the service from robotic scanners. Manually evading this layer, our live, PyShark-based detection script immediately detected the malicious traffic, providing a timely and actionable notification to the system administrator. This is an immense advantage over traditional log-based systems that are necessarily reactive. The rate-limiting rules activation of the follow-up IPTables was the most decisive throttling element, which reduced the attack and made it ineffective. Finally, Fail2Ban implemented the last countermeasure, which prohibited the attacker's IP address permanently.

This hybrid approach eliminates the limitation of point solutions. As shown in Table I, our system has faster detection than traditional log-based systems and avoids the heavy computational cost of ML-based IDS systems. It also improves Port Knocking security by adding robust reactive mechanisms. The integration of these layers results in a stronger defense posture.

One of the most important aspects of our evaluation was ensuring that the security features did not compromise system performance. The memory consumption and CPU utilization of the Python script and knockd service were minimal, with no noticeable. This demonstrates that our solution is not only functional but resource-efficient and production-capable.

TABLE I. COMPARATIVE ANALYSIS OF SSH BRUTE-FORCE ATTACK MITIGATION SOLUTIONS

Feature	Fail2Ban	Port Knocking	Proposed Multi-Layered Framework
Detection & Response	Reactive (log analysis)	Proactive (obfuscation)	Hybrid (real-time packet analysis + log analysis)
Attack Surface Reduction	None	High (Hides SSH port)	High (Hides SSH port)
Response Time	Delayed (after N failed attempts)	Instant (for obfuscation)	Instant (packet detection) + Delayed (log analysis)
Vulnerability to "Low-and-Slow" Attacks	High (if below threshold)	Low (if port is hidden)	Low (due to real-time packet analysis)
Resource Overhead	Low	Low	Low
Required Expertise	Moderate	Moderate	Low-Moderate

VI. CONCLUSION AND FUTURE WORK

This study demonstrates that by the intelligent integration of available technologies, we can create an SSH

defense system much more powerful than any one solution. Our multi-layered architecture—grounded in proactive, real-time, and reactive controls—performs outstandingly well against brute-force attacks. We end up with a practical, scalable, and easy-to-deploy security platform that robustly protects one of the most critical services on any server.

Our study has interesting possibilities for future work. For instance, we could make the detection script automatically generate firewall rules, enabling a completely adaptive and autonomous response system. And replacing fixed thresholds with an extremely lightweight machine learning model may enable more sophisticated, "low-and-slow" attack vector detection. Lastly, building an easy web dashboard to filter and graph alerts from all components would make this powerful collection of tools into one unified, easy-to-use security suite.

REFERENCES

- [1] J. Vykopal, T. Plesnik, and P. Minarik, "Network-based dictionary attack detection," in *Proc. International Conference on Future Networks (ICFN)*, Bangkok, Thailand, Mar. 2009, pp. 248–253.
- [2] J. Piet, A. Sharma, V. Paxson, and D. Wagner, "Network detection of interactive SSH impostors using deep learning," in *Proc. 32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA, USA, Aug. 2023, pp. 1157–1174.
- [3] A. AbdelRahman, D. Barrera, and P. C. van Oorschot, "What lies beneath? Analyzing automated SSH bruteforce attacks," in *Proc. Passwords Conference*, Cambridge, UK, 2015.
- [4] R. Mukherjee, "irpSSHa: Identifying and reporting SSH brute force attackers," *SSRN Electron. J.*, Jan. 2024, doi: 10.2139/ssrn.5038715.
- [5] R. George and E. Zeratsion, "Detection of SSH brute-force attacks using machine learning: A comparative study with Fail2Ban and PAM Tally2," M.Sc. thesis, Dept. of Engineering Science, University West, Trollhättan, Sweden, Jul. 2025.
- [6] C. Kelly, N. Pitropakis, A. Mylonas, S. McKeown, and W. J. Buchanan, "A comparative analysis of honeypots on different cloud platforms," *Sensors*, vol. 21, no. 7, p. 2433, 2021.
- [7] P. M. Chen, A. W. Zomaya, T. Kwon, R. K. Iyer, and Y. Wu, "Mining threat intelligence from billion-scale SSH brute-force attacks," in *Proc. 2020 Workshop on Decentralized IoT Systems and Security (DISS)*, San Diego, CA, USA, Apr. 2020.
- [8] W. Yao, L. Hu, Y. Hou, and X. L. Li, "A lightweight intelligent network intrusion detection system using one-class autoencoder and ensemble learning for IoT," *Sensors*, vol. 23, no. 8, p. 4141, 2023.
- [9] S. Jeanquier, "An analysis of port knocking and single packet authorization," M.Sc. thesis, Dept. of Information Security, Royal Holloway, University of London, London, UK, 2006.
- [10] M. Shiraz, L. Boroumand, A. Gani, and S. Khan, "An improved port knocking authentication framework for mobile cloud computing," *Malaysian Journal of Computer Science*, vol. 32, no. 4, pp. 293–310, 2019.
- [11] S. K. Wanjau, G. M. Wambugu, and G. N. Kamau, "SSH-brute force attack detection model based on deep learning," *International Journal of Computer Applications Technology and Research*, vol. 10, no. 1, pp. 42–50, 2021.
- [12] M. M. Raikar and S. M. Meena, "SSH brute force attack mitigation in Internet of Things (IoT) network: An edge device security measure," in *Proc. 2021 2nd Int. Conf. Secure Cyber Comput. Commun. (ICSCCC)*, Jalandhar, India, May 2021, doi: 10.1109/ICSCCC51823.2021.9478131.
- [13] T. Ernawati, I. Kholid, and D. Rohmayani, "Case study in network security system using random port knocking method on the principles of availability, confidentiality and integrity," *Jurnal Online Informatika*, vol. 9, no. 1, pp. 41–51, 2024.