Service Obfuscation Techniques for SSH

1st Hussein Missilmani
ham65@mail.aub.edu

2nd Mohamad Seif El Dine
mws31@mail.aub.edu

3rd Sarjoun Radiyeh
smr16@mail.aub.edu

4th Alaa Khanafer
afk13@mail.aub.edu

***Abstract*—In an era where cyber threats are increasingly sophisticated, securing network services such as Secure Shell (SSH) is paramount.**

This paper investigates how service obfuscation techniques enhance SSH security by reducing its visibility to attackers.

First, port obfuscation involves changing the default SSH port (22) to a non-standard port, complicating reconnaissance by tools like Nmap and Angry IP Scanner. This reduces the predictability of service ports, making it harder for attackers to identify and exploit SSH.

We also explore advanced methods like port knocking and Single Packet Authorization (SPA), which add security by requiring pre-authentication. Additionally, SSL/TLS tunneling is discussed, showing how SSH traffic within HTTPS disguises it from scans.

Practical experiments were conducted for each method, showing their effectiveness. These findings highlight the importance of such measures in protecting critical services, defending against unauthorized access, and increasing security alerts for companies.

I. Introduction

Services running on known ports such as SSH on port 22, are common targets for attackers using port scanning tools like Nmap and Angry IP Scanner. These tools allow attackers to find open ports and the services running on them, leading to possible exploitation. This predictability is why we need techniques to make these services harder to detect while maintaining access for legitimate users.

Our project will investigate the effectiveness of port obfuscation in reducing the visibility of services on known ports. Using SSH as a test case, we will evaluate how moving it from its default port to a non-standard port affects detectability. In addition to port obfuscation, we will explore methods such as port knocking, Single Packet Authorization (SPA), and SSL/TLS tunneling, which add layers of security by masking the service or requiring pre-authentication for access.

We will test these techniques in a controlled environment against several port scanning tools to compare their effectiveness. The study will also examine the trade-offs involved such as usability and complexity to determine the practicality of these approaches in real world scenarios. Our goal is to provide insights into improving service security in networks.

II. Literature Review

A. Port Obfuscation

Port obfuscation improves security on the network by changing standard communication patterns on specific ports to become as invisible as possible, such as running SSH on non-standard ports from the default. This ”security through obscurity” helps against basic attacks and minimizes reconnaissance using tools such as Nmap scans, but is less effective against skilled attackers using advanced strategies [1] [2]. Such attackers can also use explicit port details in URLs to bypass defenses, therefore making dynamic obfuscation a must [3]. Systems like CHAOS are using dynamic port obfuscation within a moving target defense framework, therefore trying to increase unpredictability and reduce attack surfaces [1]. However, the effectiveness of port obfuscation is maximized when integrated with layered security measures rather than being dependent on it as an independent solution [2].

B. Port Knocking

Port knocking is a security technique designed to protect services by keeping all ports closed and hidden until a specific sequence of ”knocks” is performed by an authorized user. This method provides an extra layer of authentication making it harder for attackers to detect or exploit services. According to [4], one advantage of port knocking is its ability to make servers invisible to port scans with reducing the information available to attackers. This invisibility prevents unauthorized access while also adding a layer of authentication.

It works by configuring firewalls to block all incoming connections to specific ports and requiring users to perform a predefined knocking sequence to temporarily open them, as explained in [5]. This makes sure only those with the correct pattern can access hidden services. Additionally, as noted in [6], port knocking adds security by requiring users to also authenticate with the underlying service such as SSH, even after completing the knocking sequence.

C. Single Packet Authorization

Single Packet Authorization (SPA) is an advanced Network Security technique used to increase the protection of the devices within the network, where it asks for an encrypted authentication packet before giving access to any service [10]. SPA keeps all ports closed, so the server will be hidden to the unauthorized users. Compared to port Knocking, SPA uses a single encrypted packet to authenticate access, so it prevents vulnerabilities such as unencrypted packet sequences that can be intercepted and exploited. For example, when a user tries to access a service, a packet containing a cryptographic signature is sent to the target. The system receiving the packet validates the signature, checks the timestamp to ensure the packet is not a replay, and only if the packet is verified, it grants access to the requested service, such as SSH or VPN [10].

All in all, SPA is really useful in environments where security is a must, it is an essential tool in the modern Network Security, where it provides an efficient and secure method for controlling access to sensitive services without exposing them to potential threats.

D. SSL/TLS Tunneling

SSL/TLS tunneling is a technique used for masking services by encapsulating their traffic using SSL/TLS encryption, making the service's traffic appear as just standard HTTPS traffic. This method helps services bypass firewalls and avoid detection by port scanning or packet analysis tools. As explained in [8], this approach enhances security through it's encryption and authentication mechanisms that make it significantly harder and more complex for attackers to identify or exploit SSH services running on the network.

Implementing this method usually involves configuring the server to route all incoming traffic on port 443 (HTTPS) through an SSL/TLS tunnel to the SSH service running on the server. This specific setup is done so that port scans are only able to detect an active HTTPS service while also making unauthorized users without proper certificates unable to access the SSH server listening locally. As noted in [9], though this method provides a useful amount of security and privacy, it can easily increase the system's complexity. Despite this inconvenient trade-off, this method can be relatively effective in high-risk environments where critical services must be concealed while remaining accesible.

III. Theory/Design/Methodology

A. Port Obfuscation

Our goal here is to implement port obfuscation on a target machine running SSH. We used for this experiment to Linux VM's one as a host and another as a target, changing the default SSH port (22) to a custom port (5300) on the target side. First, we modify the SSH configuration file, located at "/etc/ssh/sshd_config". The Port directive in the configuration file was changed then we save the file and restart SSH service using the command sudo systemctl restart sshd to redirect SSH traffic to the new port.

Next, concerning the firewall on the target machine we will block all incoming connections except those on port 5300. This was achieved using the ufw - Uncomplicated Firewall utility, which was used to deny access to all other ports while explicitly allowing traffic on port 5300. This ensures that the target machine was effectively invisible to port scans attempting to access SSH on the default port (22), while still allowing legitimate access to the SSH service on port 5300.

B. Port Knocking With Port Obfuscation

In this setup, we set up two virtual machines: a Linux server and a Kali Linux client both on the same private network. On the server, SSH is configured to run on a non-standard port 54214 instead of port 22, as a way to make the service harder to detect. Port knocking is implemented on the server using the knockd tool to control access to the obfuscated SSH port.

The server is set to keep port 54214 in a default "dropped" state, preventing it from being detected or accessed without authentication. A knocking sequence of 7000 8000 9000 is used to temporarily allow access to the obfuscated port. When the correct knock sequence was received, knockd is configured to add an iptables rule to allow SSH connections only from the client's IP address. After completing the session, the client would send the reverse knock sequence 9000 8000 7000 to close the port and restore it to its default "dropped" state.

C. Single Packet Authorization

For this part, we set up two virtual machines: an Ubuntu server and a Kali Linux client, connected to the same private network. On the server, SSH is configured to run on the default port 22, while access is controlled using Single Packet Authorization (SPA) with the fwknop tool.

The server is set to keep port 22 in a default "dropped" state, making it invisible to unauthorized users. The fwknopd service listens for specially crafted SPA packets containing a pre-shared key. Upon receiving a valid SPA packet, fwknop dynamically modifies iptables to allow SSH connections only from the client's IP address for a limited time. After the session ends, the iptables rules go back to the default setting DROP which denies any further connection to the server through port 22. On the client side, fwknop sends an SPA packet with the right encryption key granting temporary access to the server.

D. SSL/TLS Tunneling

The objective here is to establish the efficiency of using SSL/TLS tunneling as a method for hiding SSH traffic by encapsulating it within HTTPS traffic. Performing this experiment required setting up two VMs that are Linux-based: one as the client and the second as the target server, so that network scans originating from the outside could not identify SSH on the server but would instead indicate the presence of traffic consistent with HTTPS on port 443.

To be able to achieve this, we installed stunnel, a proxy tool that adds SSL/TLS encryption, on both our client and server machines. On the server side, we configured the stunnel.conf file such that it would establish an SSL listener on port 443 and forward any traffic to the SSH service running on localhost:22. The SSH daemon was configured to only be able to listen to the localhost interface, preventing any attempts of outside access. To allow this configuration to work, the firewall needed to be configured to only allow traffic to ports 443 (from anywhere) and 22 (from localhost only)

On the client side, stunnel was configured to connect to the server's port 443 to establish a SSL/TLS tunnel. Only clients with the correct self-signed SSL certificates (generated by the server and signed by a common CA using the openssl tool) could initiate a valid connection, which effectively blocks all unauthorized access attempts. Nmap and Angry IP Scanner were then used to perform scans against the server to check whether SSH was detectable or not under our newly setup configuration, while Wireshark was used to analyze network traffic to confirm that SSH packets appeared as ordinary HTTPS traffic.

IV. Experimental Results and Analysis

A. Port Obfuscation

Let's now dive into the effectiveness of the port obfuscation. First, an Nmap scan was performed from the host machine,

scanning the first 1000 ports on the target machine. The results showed that port 22, the default SSH port, was not open, indicating that the port had been successfully obfuscated.

After changing the SSH port to 5300 and configuring the firewall to block other ports, a second Nmap scan was performed, this time scanning the first 6000 ports. The results confirmed that only port 5300 was open, meaning the SSH service was now running on the obfuscated port and could not be accessed through the default port so our experiment was implemented successfully.

Finally, an SSH connection attempt was made from the host machine to the target machine using port 5300. The connection was successful, and the system prompted for the password of the target machine's user account. Once the correct password was entered, the SSH session was established, confirming that the port obfuscation was working as expected. This testing phase demonstrated that the SSH service was effectively hidden from unauthorized users and accessible only through the newly configured port.

B. Port Knocking With Port Obfuscation

In this setup, SSH was configured to run on port 54214 instead of the default port 22 to make the service less predictable and harder to detect. Before sending the knocking sequence a default Nmap scan did not show the obfuscated port. Using the -p- option to scan all ports the port appeared as "filtered" but the service was not identified. Running an Nmap scan with service detection -sV also failed to identify SSH on the port.

The port knocking process used a sequence of 7000 8000 9000 to open the port. Once the sequence was sent the port became "open" allowing SSH access with password authentication. After the session, the reverse knock sequence 9000 8000 7000 was used to close the port returning it to its default "filtered" state. Further SSH connection attempts failed as the port was no longer accessible.

Results showed that combining port knocking with port obfuscation improves security even more. By running SSH on a non-standard port it becomes even harder for attackers to detect the service. Less experienced attackers often referred to as "script kiddies" may fail to locate the port entirely as they might not use advanced scanning flags or techniques. Even if the port is found it will remain filtered without the correct knock sequence, adding another layer of authentication. This combined technique ensures that only users with precise knowledge of the port and sequence can access the service, making it an effective mechanism against unauthorized access.

C. Single Packet Authorization

With fwknop, connection to the SSH server on port 22 was limited to Single Packet Authorization (SPA). Initially, port 22 was initially set to be dropped, making it invisible to unauthorized users. The port scans performed by Nmap showed that the port appeared 'filtered', and no service could be identified on it. When a valid SPA packet containing a pre-shared encryption key was sent from the client, the server temporarily updated its firewall rules to allow SSH connections from the client's IP address. SSH access was successfully established, verified by system commands on the server.

If the session ends or a time period elapses, the firewall rule is deleted to restore port 22 to the default 'dropped' status and continues to block it. This approach effectively masked the SSH service and ensured that only users with the correct SPA packet could access it, significantly enhancing security.

D. SSL/TLS Tunneling

A before-and-after comparison was done to evaluate the affect SSL/TLS tunneling had on concealing SSH traffic. Initially, OpenSSH had the default configuration of the service running on the standard port 22 which led to an Nmap service detection scan easily identifying the service as SSH along with it's corresponding version. Similarly, Angry IP Scanner was able to flag port 22 as open.

After setting up the SSL/TLS tunnel configuration, an Nmap service detection scan was only able to reveal that port 443 was open and an attempt was made to guess the service as "ssl/https?" with no information on it's version. Angry IP Scanner was only able to detect port 443 as open but again, provided no information to reveal the true nature of the service.

Further packet analysis with Wireshark during a live SSH session showed that the captured traffic appeared as nothing more than standard HTTPS traffic with no visible indicators pointing to the fact that this is SSH communication.

The results show that as long as the SSL/TLS encryption remains intact, the tunneled traffic can effectively disguise itself as standard HTTPS traffic. All tools used, which are commonly used for reconnaissance and traffic analysis, were unable to reveal any distinguishing detail about the SSH service or its traffic. While this method is very effective in disguising SSH, it is relatively complex and requires very careful setting up. Nonetheless, the experiment shows the significant advantage of using SSL/TLS tunneling to enhance the security and privacy of critical services like SSH.

V. Conclusion

In our study, the experiments we performed were able to confirm that port obfuscation conceals services effectively to inexperienced attackers, while port knocking adds an extra authentication layer that protects against unauthorized detection and access. SPA is able to strengthen access control with cryptographic protocols, and SSL/TLS tunneling disguises SSH traffic as HTTPS, making it almost undetectable. Each of these methods has trade-offs in complexity, security and usability, with the more advanced techniques clearly providing a greater amount of protection for the service.

The research we conducted shows the importance of combining different techniques with standard security measures to properly secure our network's defense. Ideally, our future work could explore the possibility of automating the implementation of these solutions to allow for even the advanced techniques we mentioned to be practical in a professional environment.

REFERENCES

[1] Shi, Y., Zhang, H., Wang, J., Xiao, F., Huang, J., Zha, D., Hu, H., Yan, F., & Zhao, B. (2017). CHAOS: an SDN-Based moving target defense system. Security and Communication Networks, 2017, 1–11. https://doi.org/10.1155/2017/3659167

[2] Villanueva, J. C. (2024, May 31). Port Confusion - Is security through obscurity bad? — JSCAPE. JSCAPE. https://www.jscape.com/blog/using-nonstandard-ports-is-security-through-obscurity-really-bad

[3] URL and Domain Obfuscation Techniques - Prevalence and Trends Observed on Phishing Data. (2024, January 25). IEEE Conference Publication — IEEE Xplore. https://ieeexplore.ieee.org/document/10432841

[4] P. Lunsford and E. Wright, "Closed port authentication with port knocking," in *2005 Annual Conference*, Jun. 2005, pp. 10–309.

[5] M. Nur, "The Effectiveness of the Port Knocking Method in Computer Security," *International Journal of Information Security*, vol. 2, no. 6, pp. 861–868, Jun. 2023.

[6] M. Ü. Demircioğlu, "Analysis of port knocking mechanism," 2009.

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[8] J. Chen, F. Miao and Q. Wang, "SSL/TLS-based Secure Tunnel Gateway System Design and Implementation," 2007 International Workshop on Anti-Counterfeiting, Security and Identification (ASID), Xizmen, China, 2007, pp. 258-261, doi: 10.1109/IWASID.2007.373739.

[9] C. V. Wright, F. Monrose, and G. M. Masson, "Detection of encrypted tunnels across network boundaries," 2006 IEEE Conference on Communications and Network Security (CNS), 2006, pp. 1–9. DOI: 10.1109/SECURECOMM.2006.297660.

[10] Brar, S.S., 2021. Additional Security Mechanism in Single Packet Authorization.

DIVISION OF WORK

- Hussein Missilmani: Introduction, Port Knocking's literature review, methodology, and analysis.
- Mohamad Seif El Dine: Abstract, Single Packet Authorization's literature review, Port Obfuscation's methodology and analysis.
- Sarjoun Radiyeh: SSL/TLS Tunneling's literature review, methodology, and analysis, in addition to the Conclusion.
- Alaa Khanafer: Port Obfuscation's literature review, SPA's methodology and analysis.

YOUTUBE VIDEO DEMO LINK

https://youtu.be/s0wWYwjPBpQ