Hiding a Backdoor with Single Packet Authorization

0xLAITH

Abstract

This paper presents an implementation of Single Packet Authorization for maintaining access to a computer host. The goal of this paper is to serve as a proof of concept and to better understand ways in which SPA can be used to protect services. A prototype was developed that uses a daemon to perform all SPA tasks and sends a reverse shell to a successful SPA request. The full project code can be found on github (https://github.com/0xLAITH/SPA_Backdoor). The full specifications, technical details and drawbacks of this project are discussed throughout this paper.

Hiding a Backdoor with Single Packet Authorization

Port Knocking originated in 2003 and was created by Martin Krzywinski. Port knocking was the first networking technique of its kind that hid closed ports behind an authorization process more commonly referred to as a firewall authentication mechanism. This process consisted, in simple terms, of sending packets to a list of closed ports in a particular order. The server will then respond with a TCP handshake on the requested service. Port Knocking is one of the few techniques used to protect services from 0day vulnerabilities.

Single Packet Authorization (SPA) was created by MadHat Unspecific and Simple Nomad who presented their new protocol at BlackHat 2005. SPA was an attempt at simplifying and hardening the Port Knocking protocol by encoding all authorization information into one packet. Basic forms of SPA embed the hash of the concatenated string of a timestamp, IP and password into a UDP or ICMP packet. The receiving server would recalculate and validate this hash with its local time, IP header of the packet and the hard coded symmetric password. If it is a match, an expiring entry is made into the systems firewall granting the IP access to the service.

Port Knocking and SPA are both techniques that utilize the closed ports for service authorization. Since both of these techniques utilize daemons and not open ports, detecting the hosted services on a server utilizing these techniques is not feasible with port scanning techniques. Given the practicality and obfuscating nature of SPA, how can it be used in a penetration testing red team environment where maintaining access to a system is an ultimate goal? In other words, how can SPA be used to hide a backdoor port to the network it resides in?

Literature Review

Jeanquier, S. (2006) wrote a thesis on the technical challenges and differences between Port Knocking and Single Packet Authorization. Besides the technical details on how these two protocols defer, the limitations and practical implementations of SPA and port knocking are also discussed. Various attacks exist against both of these protocols: brute force, attacking the daemon, eavesdropping and replay attacks, man in the middle, and detection techniques. Brute forcing a port knocking backed service is a lot easier than brute forcing SPA. This attack would consist of guessing the sequence of port knocks given that the result space is finite (1 to 65535 different ports). However, such an attack is noisy on a network and therefore easily detectable. Attacking the daemon is another attack, however, this time is mostly only pertinent to SPA. The two types of daemons are log readers (for port knocking) and packet sniffers (for SPA). Log Readers work at the lower levels of the OSI model making them difficult to exploit. However, packet sniffers utilize APIs such as libpcap that could contain vulnerabilities. Due to the open source nature of libpcap, these vulnerabilities have very low probability. The interceptions and impersonation attacks are mostly a problem for port knocking as they are susceptible to eavesdropping unlike SPA which has its authorization data obfuscated. Jeanquier concludes that while SPA seems to have inherent advantages both reduce the amount of threats to the system they reside in.

Rash, M. (2006) is at the forefront of SPA technology. Rash is the developer of FWKNOP which is an open source SPA service provider. In his white paper, Rash displays the technical details of the FWKNOP project. While the project has advanced to provide SPA to a variety of services, the original white paper focused on securing openSSH behind SPA. The

following data is sent to the knock server to authenticate a host: initialization vector, username,

timestamp, fwknop version, mode (access or command), desired access and a MD5 sum. The IV

serves to prevent replay attacks as a log is maintained on the knock server. The version number

provides backwards compatibility. The MD5 value serves as a message integrity checking

mechanism. The whole message is encrypted using RSA and concatenated with semicolon

characters. Once a successful knock packet is sniffed using the libpcap library, a 10 second

expiring firewall rule is created for the IP of the knock packet for openSSH.

Given these pieces of research, it is clear that SPA is the better port obfuscation technique

according to Jeanquier. Additionally, the details of an existing implementation of SPA can be

learnt from Rash's white paper of FWKNOP. All of the information needed to construct a SPA

solution to a hidden reverse shell backdoor are gathered. Given that no solutions currently exist

for such a problem a solution will be made from scratch using libpcap.

Procedures

The backbone of any SPA service is packet sniffing technology. This project will base its

core technology on the ones used in the FWKNOP project. Libpcap will be used as it is widely

regarded as the best open source repository for packet sniffing. The choice of encryption

technology will be the golden standard for symmetric encryption: RSA (in ctr mode). Another

open source project will be used to provide RSA encryption to the solution: LibTomCrypt**.**

Lastly, a client will need to also be developed to construct the appropriate packets that will be

read by our SPA server. While such an application is possible to develop in C, a short python

script will be used instead as the Python Scapy API allows the construction of a TCP/IP packet

in a single line of code. It is important to note that simple tools like netcat will not work for SPA
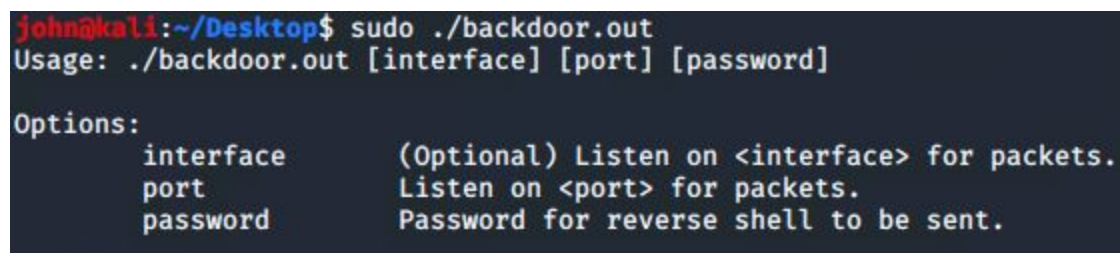
as these tools perform a TCP handshake before sending any payload data, therefore a custom

packet constructing/sending tool has to be developed.

In order to generate a reverse shell two main data points are obviously needed: a remote

IP and port. Therefore, the SPA packet will have to contain these two pieces of information. You

may note that the IP could just be read from the header of the TCP packet. However, the choice

to embed the actual IP and port in the payload of the packet will prevent replay attacks. If the

SPA server only sends reverse shells to embedded IPs then only those with the credentials to

encrypt the payload can receive a reverse shell. Therefore, a replay attack will simply direct a

reverse shell to the original sender of the packet.

Similarly to the FWKNOP project, the choice to embed a timestamp within the SPA

packet is also followed. This adds an additional layer of security to our backdoor as every

generated SPA packet has a time to live of 10 seconds further preventing replay attacks.

Results

The final solution contains 3 key files: backdoor daemon, the packet sender and the

encrypter. The backdoor daemon is recides on the SPA host and acts as a packet sniffer awaiting

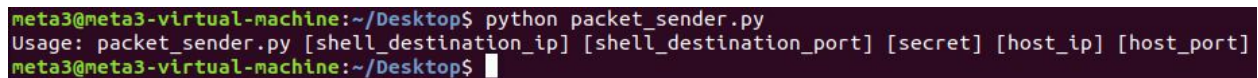a SPA packet on a particular port. See the below figure for its usage:

Figure 1 - *Backdoor Daemon Usage*

The backdoor daemon is launched on a particular interface, port number and with a secret password used in the RSA encryption (64-bit). The port number signifies the port to sniff packets in order to minimize the load of the packet sniffer and add a layer of obfuscation to the backdoor as any port can be selected. The daemon can run in the background continuously as it sniffs and reads all packets on the particular designated port number. When it receives a packet, it decrypts the payload (if any), and attempts to validate any requests. A successful SPA connection does not close the daemon.

The packet sender utility is written in python and utilizes a C program to encrypt the payload of the SPA packets. A C program is used separately for the encryption to maintain consistency between the used cryptographic functions of different libraries. The encrypter C program also uses the LibTomCrypt API to implement the RSA CTR encryption. The packet sender is configured to send a TCP SYN packet with an encrypted payload. See the below figure for the packet sending script usage:

```
meta3@meta3-virtual-machine:~/Desktop$ python packet_sender.py
Usage: packet_sender.py [shell_destination_ip] [shell_destination_port] [secret] [host_ip] [host_port]
meta3@meta3-virtual-machine:~/Desktop$ 
```

Figure 2 - *Packet Sending Client Usage*

The usage of the packet sending client is quite self explanatory. It requires the specification of the SPA host's ip and port and the target IP and port of the reserse shell alongside the password. It is important to note that the password has to be 16 characters long (64-bit) for this to work. Here is the list of items included in the payload:

- Initialisation Vector
- Timestamp (epoch time)

- IP address

- Port Number

All of these items are concatenated with semicolons. Everything but the IV is encrypted

with the shared password. See figures below of the SPA backdoor in action:

Figure 3 - *Packet Sender Script Running*

Figure 4 - *SPA Backdoor Daemon Running*

Figure 5 - *Reverse Shell Received*

These programs were designed for linux systems. In the above figures, the remote host is

running Ubuntu 18, and the SPA host is running Kali 2020. Here is the process to reproduce

these results:

- Run all programs on Linux Systems

- Install dependencies on both systems:

    - On SPA system:

        - `sudo apt-get install libtomcrypt-dev`

        - `sudo apt-get install libpcap-dev`

    - On Remote system:

        - `sudo apt-get install python`

        - `sudo apt-get install libtomcrypt-dev`

        - `pip install --pre scapy[basic]`

- Compile backdoor daemon on SPA system:

    - `gcc backdoor.c -lpcap -ltomcrypt -o backdoor.out`

- Compile encrypter C program on Remote system:

    - `gcc encrypter.c -o encrypter -ltomcrypt`

- Run SPA Daemon:

    - `sudo ./backdoor.out [interface] [port] [password]`

- Open listening port on remote system to receive reverse shell using netcat:

    - `nc -lvnp [remote_port]`

- Send SPA Packet to SPA Daemon from Remote System from a different shell.

    Note: must have encrypter compiled in the same directory as the python script.

```
o  sudo python packet_sender.py [remote_ip]

   [remote_port] [password] [spa_ip] [spa_port]
```

- A reverse shell will appear in the netcat terminal window.

Conclusion

The current project definitely has its drawbacks. For instance, the reverse shell is unencrypted, the RSA key size is only 64-bit, and the code is not professionally written. The code definitely has its own flaws as not all inputs are sanitised. The choice to embed data in a TCP SYN packet was simply one of convenience. Technically any packet could be used to embed data. An analysis of a better protocol to embed the data in should be made. Given that TCP SYN packets are not designed to have payloads, these packets would get caught by any good external firewall. However, as an initial proof of concept, this is a successful implementation of SPA.

This project can be expanded upon in many ways. For starters, the SPA daemon is only able to run on ethernet interfaces as it currently is only able to parse 802.1Q headers. While this would suffice for most systems, it may be interesting to expand this to work with 802.11 headers for hosts with exclusive access to Wi-Fi networks. Additionally, while the dependencies of the project are slim, they could always be made smaller. Expanding the platforms (e.g. Windows) for which these scripts can be run on is also an interesting task. Lastly, one could take the obfuscation of the backdoor even further by attempting to hide the daemon in memory to not only hide it from the network, but also the system it is running on.

Overall, a solution that utilizes the obscurity of SPA to hide a reverse shell backdoor on a system was accomplished. As a result, the backdoor can only be detected within the system it is running on. SPA is a very powerful tool that is currently underutilized in the cybersecurity space. Hopefully this paper demonstrated just one of the many ways one can utilize the potential of SPA.

References

Jeanquier, S. (2006). *An Analysis of Port Knocking and Single Packet Authorization MSc Thesis* (Doctoral dissertation, PhD thesis).

Rash, M. (2007). Protecting SSH servers with single packet authorization. *The Linux Journal*, *2007*(157).

Rash, M. (2006). Single packet authorization with fwknop. *login: The USENIX Magazine*, *31*(1), 63-69.

Smits, R., Jain, D., Pidcock, S., Goldberg, I., & Hengartner, U. (2011, October). BridgeSPA: Improving Tor bridges with single packet authorization. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society* (pp. 93-102).

Tariq, M., Baig, M. S., & Saeed, M. T. (2008, July). Associating the Authentication and Connection Establishment Phases in Passive Authorization Techniques. In *Proceedings of* *the world Congress on Engineering* (Vol. 1).