A new approach to building a disguised server using the honey port against general scanning attacks

Hyun Soo Park, Young Bae Jeon, Ji Won Yoon

Abstract The port scan is a well-known technique which malicious people often use before attacking a server. The attackers obtain the fingerprint of the target server by scanning ports and then make an attack scenario. Several approaches including the 'port knocking' and 'Single Packet Authorization' (SPA) have been developed to defense port scanning attack and allow only authenticated users to access ports. However, the approaches have a disadvantage that the attacker can obtain the information about the ports by applying inference techniques given observed patterns. If a router, connecting the server to the outside, is cracked by the attacker, he or she could infer particular ports which authenticated users consistently use to communicate with the server. In this paper, we propose a new defense method, *Honeyport*, which can prevent the attackers from obtaining the information about ports and make them demotivated by disguising the server as peripherals. Furthermore, by adopting packet encryption as in IPSec, the attacker cannot obtain the critical information via packet sniffing in our proposed model.

1 Introduction

Port scanning is one of the major and crucial functions to identify and connect devices for adequate communication in a network. However, this function can be abused by the malicious people, unlike the original purpose. Attackers obtain the fingerprint of a victim server by scanning the server's ports to find the vulnerability and weakness of the victim server. With the scanned fingerprint, the attacker can just check which ports are opened and which services are provided. Afterward, the attacker explores the adequate attack scenario to give a damage the target server or to compromise the victim server.

Hyun Soo Park, Young Bae Jeon, and Ji Won Yoon
Center for Information Security Technologies(CIST), Korea University, Republic of Korea,
Corresponding author is J. W. Yoon e-mail: jiwon_yoon@korea.ac.kr

L. Barolli et al. (eds.), *Advances on Broad-Band Wireless Computing, Communication and Applications*, Lecture Notes on Data Engineering and Communications Technologies 2, DOI 10.1007/978-3-319-49106-6_44

That is, since the attacker can find the proper attack types using the port scan, we need to develop a protection system which removes or avoids such unwanted port scanning attack. Several defense approaches including the port knocking[3] and the Single Packet Authorization(SPA)[4, 5] have been developed to hide the information about the server's ports. They are designed to hide the using port about when the server transmits and receives packets over the communication by the authentication. IPSec is an alternative approach which encrypts and encapsulates packets given an IPSec protocol to communicate. Although packets are sniffed from outside, the server can prevent the packet's information from being exposed by the IPSec because it is encrypted. Also, a new scheme called honeypot has been introduced to hide the server's identity. This technique installs a trap server and leads the hacker to attack the trap server, not a real server. The trap server collects and analyzes attackers' logs.

However, there are few techniques which fundamentally prevent hackers' attack trials. While ports are mainly focused in port knocking and the SPA, IPSec considers not the port but data itself using encryption. And the honeypot's point of view is the defense about the only cheating attacker. In this paper, we propose a new technique to integrate all of these conventional approaches and lead the attackers not to intrude the server fundamentally.

This paper is organized as follows. In Section 2, we describe background knowledge about the port scanning, the port knocking, and the SPA. We propose the algorithm about *honey port* in Section 3 and show the results of the experiment related to *honeyport* in Section 4. Finally, we evaluate our approach and conclude with an overall summary of our approach.

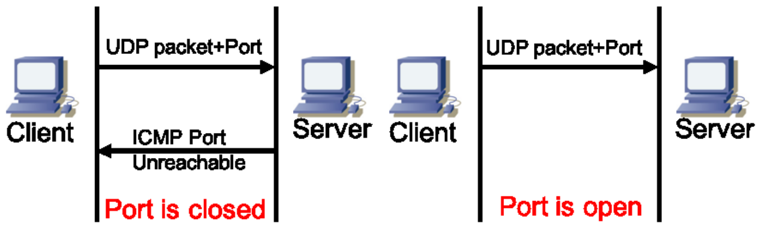
2 Background

2.1 Port Scanning

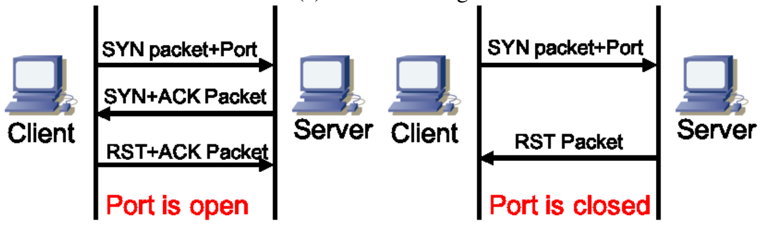
It is known that the port scan is used to scan a target system in order for exploring its potentially exploitable service. There are mainly two different ways in the port scanning: User Datagram Protocol (UDP) scanning[1, 2] and Transmission Control Protocol (TCP) scanning.

Fig. 1-(a) shows how UDP scanning works. In UDP protocal, "ICMP Port Unreachable" message is generated by the server to inform the client that the destination is unreachable if the destination port is said to be closed. Otherwise, if the client does not recieve any message from the server after sending the the UDP packet with a destination port, we may infer that the destination port of the server is opened.

However, while the client certaintly knows the closeness of the port but cannot exactly know the openness of the server's port since the response packet has been lost in UDP protocol.



(a) UDP Scanning



(b) TCP Scanning

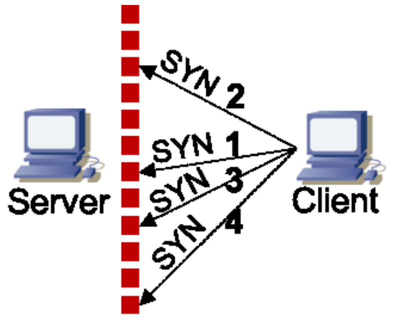
Fig. 1: UDP Scanning and TCP Scanning

TCP is the other port scanning protocol to find the opened ports [1, 2]. Figure 1-(b) shows how TCP Scanning works. When sending a SYN packet to a destination port over TCP Communication, the client receives a $SYN + ACK$ packet if the port is opened. Otherwise, the client receives a $RST + ACK$ packet. Thus, the attacker obtains information about which ports are opened.

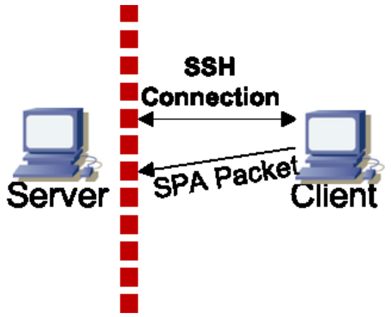
2.2 *Port Knocking*

To date, several approaches have been developed to prevent an attacker from the unwanted port-scan.

One of the most well-known approaches is the port knocking[3] which opens ports on a firewall by generating a connection attempt on a set of pre-specified closed ports. When a sequence of connection attempt is correctly received, a server authorizes a user as shown in Fig. 2-(a). Attackers can approach to the server through the firewall but they will fail to connect the server since they do not know the correct sequence of the connection attempt. However, there is vulnerability in this protocol. If the router which connects the server to the outside is compromised by an attacker, the ports are exposed to the attacker.



(a) Port Knocking



(b) Single Packet Authorization

Fig. 2: Two approaches to hide opened ports against port scanning attacks: (a) 'Port Knocking' and (b) 'Single Packet Authorization' (SPA)

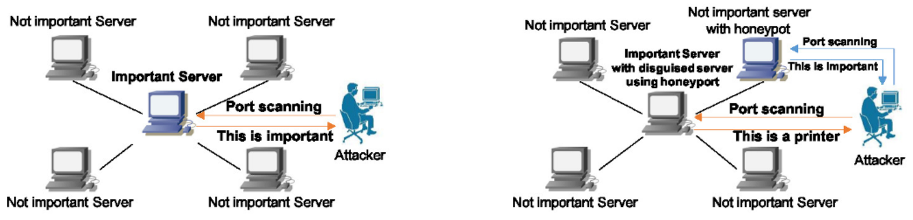
2.3 Single Packet Authorization

Single Packet Authorization(SPA)[4, 5] is a similar way to port knocking in that it requires only a single 'knock' for the communication. SPA combines packet filter via drop and packet sniffer. SPA uses packet's payload to prove the ownership of the information, not the packet's header in Fig. 2-(b). Clients send only one packet about own identity to the SPA server. This is possible because MTU's size is hundreds bytes unit in the common network. However, this approach has the same vulnerability with the port knocking.

3 Proposal Algorithm

The background section shows two main approaches to preventing port scan from the attacker: 'port knocking' and 'single packet authorization' (SPA). However, their effectiveness is rather limited, and the attacker can bypass the approaches using their vulnerabilities. For instance, when ssh is not used in the approaches, the attacker can infer which ports are used because they consistently use the identical ports. In addition, if the routers linking to the victim server are compromised by the hackers, the attacker can monitor the pattern and sequence of the 'knock's.

From this point of view, we propose an algorithm which fixes these vulnerabilities in the conventional approaches. Therefore, in order to remove the vulnerabilities, we propose a scheme to disguise the victim system using fake ports. In this paper, we are disguising the target server as a trivial terminal which looks ignorable while an actual dummy terminal is also disguised as a significant server by *honeyport*. In the proposed method, the server falsifies the response which is requested from the port scanning by the attacker and sends it to the attacker. In Figure 3, we can see that the attacker confuses the important server with the printer.



(a) Without the honeyport and the honeypot (b) With the honeyport and the honeypot

Fig. 3: Administrators can hide their property by opening fake ports and we call this by *honeyport*. The important server can be disguised as an insignificant sever using *honeyport*: we can see that the attacker regards the important server as the printer in this figure.

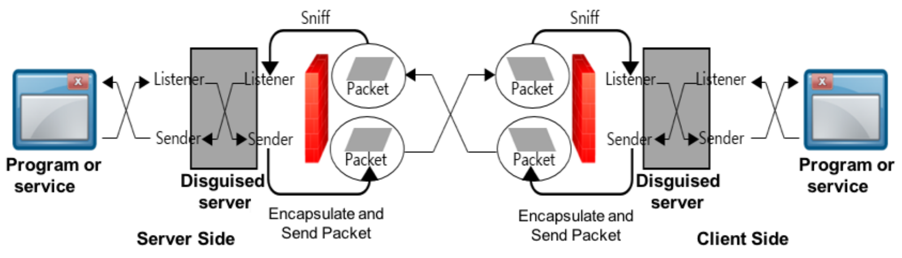


Fig. 4: Flowchart about the disguised server: The server and client communicate each other by the disguised server through *Listener* and *Sender*.

For this, a system administrator can make the disguised server as plotted in Figure 4. The disguised server's function consists of four parts : (a) *listener*, (b) *sender*, (c) *spoofer* and (d) *sniffing packets*.

1. *Listener* catches packets from outside to inside and from inside to outside;
2. *Sender* forwards packets from outside to inside and sends packets from inside to outside;
3. *Spoofer* provides fake information to attacker during port scanning. However, because used ports are closed and hidden by the firewall, programs actually cannot catch packets from outside; and
4. *Sniffing packets*, which is operated in the disguised server, collects all packets blocked by the firewall blocks because the port is closed. The disguised server brings all packets filtered by the firewall and delivers them to the listener.

3.1 Listener

Listener catches two kinds of packets, which move from inside to outside and from outside to inside.

3.1.1 Packets going from inside to outside

Listener sniffs packets going from inside to outside and sends the packets to *Sender* after the series of following processes:

1. *Listener* firstly attaches a source IP address, a source port, a destination port, and a timestamp to packet's payload.
2. *Listener* encrypts it using Advanced Encryption Standard (AES) with a shared key.
3. *Listener* attaches a hash value returned by a hash function, HMAC, which has a parameter of packet's encrypted payload.
4. *Listener* attaches ”Sniffer” to the payload to recognize when packets arrive.

After these processes, the structure of a packet's payload is shown in Figure 5. We can see the tag written as ”Sniffer” in the payload's head and MAC value behind it. Lastly, we can find out the encrypted information of IP address, Port, Timestamp and Original payload in the tail of the packet's payload. *Listener* sends these packets to outside by *Sender*.

The reason for putting additional information into the existing payload is to prepare defense against various attacks. By adding IP address, port, and timestamp into the payload, we prevent replay attack and ARP Spoofing. Besides, by adding MAC into the payload, we prevent that the attacker falsifies packets.

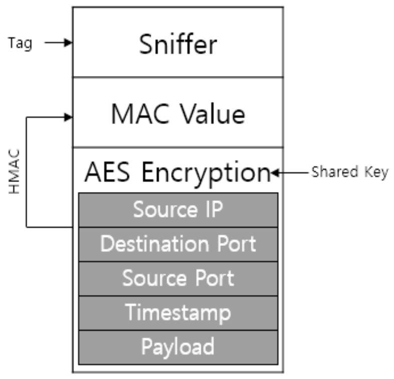


Fig. 5: Payload encapsulation of packet

3.1.2 Packets going from outside to inside

Packets going from outside to inside is processed with the inverse order of processes for packets going from inside to outside. This process is called decapsulation. Because the port which should be used is closed, packets are passed only through the disguised server. In the disguised server, there is much overhead to check every packet. Therefore, we sniff only packets which are labeled as ”Sniffer” in front of packet’s payload. This has the following steps:

1. *Listener* detaches ”Sniffer” in packet’s payload.
2. *Listener* compares MAC in packet’s payload and the value returned by a hash function, HMAC, which has a parameter of the ciphertext in payload.
3. *Listener* decrypts the ciphertext in payload and compare source IP address and own IP address in decrypted text.
4. *Listener* verifies timestamp.
5. *Listener* removes the additional information in the payload except for original payload.

After these processes, the packets are forwarded to inside. Services or programs which have received it from the disguised server can operate without changing the existing protocol since packet recover by decapsulation. The details of the *Listener* is demonstrated in Algorithm 1.

Firstly, the first **if** statement means that if destination IP address and server IP address are equal, and payload includes “Sniffer”, the packets are entered to inside from outside. After making sure the ciphertext forgery by comparing MAC, we decrypt the payload’s ciphertext with a shared key. In this process, if the ciphertext

Algorithm 1 Pseudocode for *Listener*

```
 1: procedure LISTENER(pkt)
 2:     if $pkt[IP].dst = my\_ip$ and $pkt[Raw].load$ include "Sniffer" then
 3:         if $pkt[Raw].load.MAC = \mathbf{HMAC}(pkt[Raw].load.data)$ then
 4:             Try :
 5:                 $PlainText \leftarrow \mathbf{AES\_Decrypt}(pkt[Raw].load.data, key)$
 6:             Except :
 7:                 Print "Wrong key"
 8:                 return false
 9:             if $PlainText.ip = pkt[IP].src$ and $PlainText.timestamp$ is validate then
10:                 $Forward\_pkt \leftarrow pkt$
11:                 $Forward\_pkt[Raw].load \leftarrow PlainText.payload$
12:                 $Forward\_pkt[Raw].dport \leftarrow PlainText.dport$
13:                 $Forward\_pkt[Raw].sport \leftarrow PlainText.sport$
14:                 Calculate Checksum of Forward_pkt
15:                 Sender( $Forward\_pkt$ )
16:     else if $pkt[IP].src = my\_ip$ then
17:         $pkt[Raw].load \leftarrow pkt[Raw].load|my\_ip|pkt[TCP].dport|pkt[TCP].sport|timestamp$
18:         $CipherText \leftarrow \mathbf{AES\_encrypt}(pkt[Raw].load, key)$
19:         $pkt[Raw].load \leftarrow "Sniffer"|\mathbf{HMAC}(CipherText, key)|CipherText$
20:         Sender( $pkt$ )
```

is decrypted with wrong key, print "Wrong key". After decryption, we verify the real sender by comparing IP addresses and validate the timestamp. If every process is verified, the payload is decapsulated to original payload and port is changed correctly. The packet is sent by *Sender* to inside after recalculating the checksum. When the packet's source IP address and server IP address are equal, the packets are going to outside from inside. Therefore, we encapsulate packet's payload and attach ciphertext and MAC, and send the packet to outside by *Sender*.

3.2 Sender

Sender takes packets from *Listener* and send them to outside or inside. Following subsections represent how to process the outgoing or incoming the packets in *Sender*.

3.2.1 Packets going from inside to outside

After the series of processes of a packet from *Listener*, *Sender* sends the packet to outside with a random port. Besides, while destination IP address is fixed, the destination port and the source port are set randomly. Through this process, even if the packet has been sniffed from midway, the attacker cannot infer the source port and destination port.

3.2.2 Packets going from outside to inside

Sender does not have to process the packet from outside to inside. It is because *Listener* changes the port of the packet in payload to original port. Furthermore, although the packet's source IP address is same as a public IP address, the packet is not blocked by the firewall because they are sent from an internal program, disguised server. Moreover, the service and program can use original protocol since IP address has not been changed and payload is original. For these reasons, we can simply send the packet which is sent from *Listener* to *Sender*.

We can see the pseudocode for *Sender* in Algorithm 2. As the almost every task has been performed in *Listener*, *Sender* simply changes the packet's port to a random port. If the destination IP address is same as the server IP, *Sender* sends it to loopback or internal network from inside of the firewall as *Listener* has already handled the whole work. In the other case, if the source IP address is same as the server IP address then the packet is going from inside to outside. As the packet will be delivered to outside, we should change the ports to random ports to conceal the information about the port.

Algorithm 2 Pseudocode for *Sender*

```
1: procedure SENDER(pkt)
2:     if $pkt[IP].dst = my\_ip$ then
3:         Send( $pkt$ )
4:     if $pkt[IP].src = my\_ip$ then
5:         $pkt[TCP].dport \leftarrow$ Random$(0, 65535)$
6:         $pkt[TCP].sport \leftarrow$ Random$(0, 65535)$
7:         Send( $pkt$ )
```

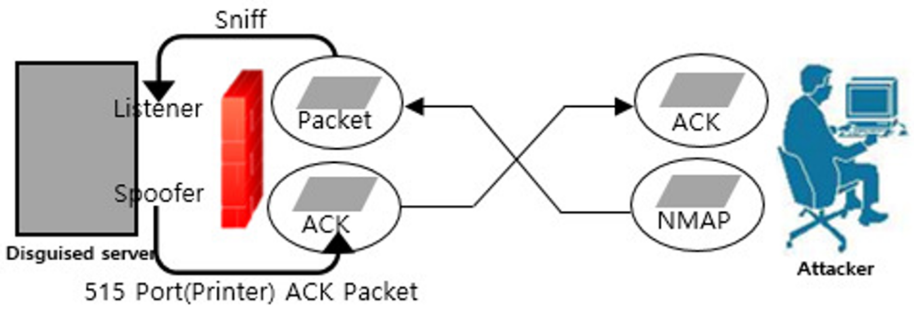


Fig. 6: Port Scanning by an attacker in our approach: If the attacker sends Nmap command to the server for obtaining the information of the server, the server responds to the attacker with ACK packet which makes the attacker regard this as the printer by *Spoofer*.

3.3 Spoofer

Spoofer is a fundamental module of this proposal. If the attackers perform the port scanning to the server and discover that all ports are closed, they will think that the server uses a security program, and then make the scenario to crack the security program. However, as we can see in Fig 6., if the attacker performs port scanning through *Spoofer*, we response with ACK packet for SYN request in 515 port which is usually used in printer. Through this way, the attacker could be led to confuse this server with the printer. As a result, the attacker do not feel the need to attack the scanned sever. In this paper, we will call the port, which response with the falsified information such as 515 port, as the *honey port*.

We can see the pseudo-code for *Spoofer* in Algorithm 3. The general way to find the open ports in port scanning is to make the TCP 3-way handshake. By using this fact, *Spoofer* performs the 3-way handshake to delude the attacker into believing that the port is open when the attacker probes whether 515 port is opened or closed by using command such as *Nmap*. By this method, we make the attacker assume that the server is the printer. To do this, if the packet is received through 515 port, the disguised server creates a new packet, and then set a property of the packet. First, we set ACK value to SEQ value plus one. Second, we set the flag to SYN/ACK.

Algorithm 3 Pseudo-code for *Spoofer*

```
1: procedure SPOOFER(pkt)
2:     if $pkt[TCP].dport = 515$ then
3:         $ACK\_pkt \leftarrow pkt$
4:         Swap( $ACK\_pkt[TCP].sport, ACK\_pkt[TCP].dport$ )
5:         Swap( $ACK\_pkt[IP].src, ACK\_pkt[IP].dst$ )
6:         $ACK\_pkt[TCP].ack \leftarrow pkt[TCP].seq + 1$
7:         $ACK\_pkt[TCP].flags \leftarrow$ SYN/ACK
8:         Calculate Checksum of ACK_pkt
9:         Send( $ACK\_pkt$ )
```

Lastly, we recalculate the checksum and send it to the opponent who performed port scanning with the port used the opponent.

We proceed the experiment with *Server(S)*, *Authenticated Client(AC)*, *Attacker(A)* to build and test our proposed algorithm. We first implement the disguised server, and execute the web server in 80 port with Python, *SimpleHTTPServer* in *Server*. For convenience sake, we will call *S*'s IP address to 1.2.3.4. In a normal condition,

4 Result

Table 1: Procedure of an experiment

```
Server> ./Disguised_server
Server> python -m SimpleHTTPServer 80
```
```
Authenticated Client>./Disguised_server
Authenticated Client>curl 1.2.3.4
<html>
<body>
Disguised Server Test
</body>
</html>
```
```
Attacker>curl 1.2.3.4
curl: (7) Failed to connect to 1.2.3.4 connection refused
Attacker>nmap 1.2.3.4 -Pn
Starting Nmap 7.01 ( https://nmap.org ) at
Nmap scan report for
Host is up (0.0062s latency).
PORT STATE SERVICE
515/tcp open printer
```

when we use *Nmap* command to the server which does not have the firewall and the disguised server, the result shows that 80 port is open. Then, *AC* also executes the disguised server and use the ‘curl’ command to access the web server in *S*. Since *A* does not have any right key for the disguised server, *A* executes the disguised server with the wrong key or execute the ‘curl’ command without the disguised server. Nonetheless, *A* uses *Nmap* command to scan ports and find out the fingerprint. If *S*, *AC* and *A* perform each command, the result is like Table 1.

Table 2: Nmap result of the actual printer

```
$>nmap 2.3.4.5 -Pn
Starting Nmap 7.01 ( https://nmap.org ) at
Nmap scan report for
Host is up (0.0062s latency).
PORT,STATE SERVICE
515/tcp open printer
631/tcp open ipp
9100/tcp open jetdirect
```

As we can see from the result, *AC* uses the right key to the disguised server and accesses to the web page without any problem, while *A* cannot. And also, even if *A* performs *Nmap* for port scanning to *S*, the result shows that only the SYN/ACK packet is received from 515 port which is usually used in the printer. Moreover, we can see that the port scan result is similar to the results which is obtained when the actual printer is scanned as shown in Table 2. For convenience sake, we set the printer IP address to 2.3.4.5. In order to show not only 515 port but also other ports such as 9100 port, we just need to modify the source code of *Spoofer* a little.

Table 3: Calculation of the packet size

Case	**Calculation**	**Bytes**	**Note**
Sniffer Tag	1 byte * 7	7 bytes	7 letters ”Sniffer”
MAC	(160bits/4bits) bytes	40 bytes	Using HMAC 160 bits. Represent one letter instead of hexadecimal
CipherText	$\lceil 256\text{bits}/6 \rceil$ + 1 byte	44 bytes	Using AES 256 bits. Dividng by 6 because of encoding Base64
Split Character	1 byte * 2	2 bytes	Split letters for Sniffer, MAC, CipherText
Sum		93 bytes	

Through this, *Attacker* cannot recognize the real web server so the attacker is highly likely to change the target. However, the proposed method in this paper essentially has overhead in encryption and decryption. In this experiment, while an average time is 0.017 seconds in the case without the disguised server, an average time is 0.0690 seconds in the case with the disguised server. The other matter is packet size. The packet which passed the channel of the disguised server contains ”Sniffer”, MAC, encrypted values and split characters so that the size is bigger than the origin. The amount of increased size is proportional to payload size, and the encapsulated packet is 93 bytes bigger than the general one without the payload. The reason why 93Bytes has been increased is in Table 3.

5 Conclusion

In this paper, we propsed a new approach to disgusing hosts using honyport and encryption against port-scanning attacks. In this proposed approach, the server uses the fake ports against the unwanted port scan attack. The attacker who scanned ports port scanning recognizes the device as a printer using honeyport. Furthermore, the packet encryption is also embeded in the approach. Therefore, the user who does not have the right key cannot get any information of the packet and is not permitted to access.

There are issues which have to be improved in this paper. Not only simply sending ACK packets of the printer’s port, but also the specification of the printer is required so that the attacker can be perfectly confused of the target device with a required so that the attacker can be perfectly confused of the target device with a printer. Also, we should confirm that program can protect the packet, and any of information is not leaked by the program. If the router is cracked by the attacker, in theory, our proposed algorithm is secure. Nevertheless, we should simulate that our proposed algorithm is secure in the condition where the router has been compromized.

References

1. De Vivo, Marco, et al. ”A review of port scanning techniques.” ACM SIGCOMM Computer Communication Review 29.2 (1999): 41-48.
2. Lyon, Gordon Fyodor. Nmap network scanning: The official Nmap project guide to network discovery and security scanning. Insecure, 2009.
3. Ali, Fakariah Hani Mohd, Rozita Yunos, and Mohd Azuan Mohamad Alias. ”Simple port knocking method: Against TCP replay attack and port scanning.”Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), 2012 International Conference on. IEEE, 2012.
4. Rash, Michael. ”Single packet authorization with fwknop.” login: The USENIX Magazine 31.1 (2006): 63-69.
5. Michael Rash (March, 2014) Single Packet Authorization with Fwknop Cipherdyn. Retrieved from http://www.cipherdyne.org/fwknop/docs/SPA.html
6. Doraswamy, Naganand, and Dan Harkins. IPSec: the new security standard for the Internet, intranets, and virtual private networks. Prentice Hall Professional, 2003.
7. Davis, Carlton R. IPSec: Securing VPNs. McGraw-Hill Professional, 2001. Ferguson, Niels, and Bruce Schneier. ”A cryptographic evaluation of IPsec.”Counterpane Internet Security, Inc 3031 (2000).
8. Provos, Niels. ”A Virtual Honeypot Framework.” USENIX Security Symposium. Vol. 173. 2004.
9. Krawetz, Neal. ”Anti-honeypot technology.” Security & Privacy, IEEE 2.1 (2004): 76-79.