Port Knocking Replay Attack Prevention with OTP S/Key and Diffie Hellman Key Exchange

M. Yusuf Bambang Setiadji, S.ST., M.Kom.
Cyber Security Engineering Study Program
Sekolah Tinggi Sandi Negara
Bogor, Indonesia
yusuf.setiadji@stsn-nci.ac.id

Muhamad Arie Taufik Rido Ganatuloh, S.Tr.TP.
Cryptography Engineering Study Program
Sekolah Tinggi Sandi Negara
Bogor, Indonesia
muhamad.arie@student.stsn-nci.ac.id

***Abstract*— Port knocking is a technique for adding an additional layer of access control before accessing certain ports on a computer. Unfortunately in the standard port knocking protocol, the knock sequence can be eavesdropped by anyone, and retransmitted/replayed. The replay of knock sequence by an adversary will omit the access control provided by port knocking protocol. In this research, we implement OTP S/Key protocol, combined with Diffie Hellman Key Exchange (DHKE), to supply a knock sequence that changes every time it is used, thus preventing knock sequence retransmission or also known as replay attacks. By comparing the intercepted port sequence of the standard port knocking protocol and the proposed protocol, it is concluded that OTP S/Key and DHKE can prevent replay attacks in port knocking.**

Computer ports; Diffie Hellman Key Exchange; OTP S/Key; replay attack; port knocking

I. INTRODUCTION

Every device connected to a network will have one or more of its port open. The opened port/s are used to communicate with other devices within the same network or to the internet. Although useful, opened ports inherits several threats to the device [1]. With the advent of various port scanning applications, a cleverly constructed data packet can be sent to an open port and render the device inoperable, because of the way it is interpreted by the software running on the device. Closing all ports is not a solution either [2], because it will diminish the intention of connecting it to a network.

The solution is to implement an additional access control layer before someone can access the port. Such access control must have the ability to alter the condition of a port, from open to close, and vice-versa. Various technique are invented to achieve said control, and the technique that are the focus of this research is the port knocking technique.

Initially, port knocking is implemented by configuring a firewall to drop all packets directed to the computer ports. If someone can authenticate itself to the firewall, then it will be granted an open port [3] [4]. The authentication takes place by accessing certain closed ports in a correct sequence, also known as "knocking". The knock are detected by the firewall, if the ports and sequence are correct, then access is granted. This condition gives an extra layer of security for the port. It is never seen open to the public, but can still be accessed if necessary.

Unfortunately, basic port knocking technique has a flaw. The ports accessed and sequence can be monitored by an adversary using a network monitoring application (e.g. Wireshark, Ethereal, tcpdump). If the ports and sequence are known, then the additional access control are useless, because anybody can repeat/replay the ports and sequence, hence defeating the purpose of the port knocking technique. Our contribution in this research is modifying the port knocking technique so it will be inherently secure against replay attack.

The rest of this paper is structured as follows. Section two will discuss about previous research in the area of port knocking. Section three consists of the methodology used in conducting this research. Section four details the design and implementation of the modified port knocking technique. In section five we will compare the modified port knocking against the original port knock implementation. Lastly, in section six, we will conclude our research.

II. RELATED WORKS

A. Simple port knocking method: Against TCP replay attack and port scanning [5]

In [5] research, the author proposes a modified port knocking technique by using the source port sequence as the authenticator, while the destination port remains unchanged. The series of knocks, with source port predefined, are delivered to an open port in the destination. The author further secures the port knocking technique by configuring the SSH to run on a custom port, rather than the usual port 22. It is argued that this implementation is faster to process than the original port knock and Fwknop+SPA [5].

The solution provided by [5] to prevent the TCP replay attack is by using the start and stop service after a successful port knock, not through the modification of the port knocking technique.

B. SKnock: Port-Knocking for Masses [6]

The focus for SKnock is scalability. The author proposes an implementation of port knocking with x509 certificates so it can be easily scaled for larger use. It argues the usage of a shared secret in the original port knocking technique hinders it from being adopted by the masses. SKnock is not without a drawback. Although it is proven to scalability, but compared with the other

ISBN 978-602-53524-0-9

port knocking technique, the payload and processing prerequisite overhead is high.

C. *Advanced port knocking authentication scheme with QRC using AES [7]*

Reference [7] research implements a complex source IP masquerading to deliver the knock ports. It also incorporates off band delivery of PRNG sequence via SMS. Although the intended purpose are achieved, which is preventing replay attack, but to accomplish it, the author sacrifices performance (by using off band) and increasing the complexity of implementation.

III. RESEARCH METHODOLOGY

This research aims on securing port knocking from replay attack. Although solutions from our literature review argues of remediating the attack, but their approach often sacrifices performance. To obtain the desired level of security and performance, we design and implement a new solution. The proposed solution will be evaluated on the basis of replay attack vulnerability compared to original implementation of port knocking. The evaluation objective is to know whether the proposed solution is susceptible to replay attack or not. Another aspect besides replay attack vulnerability is the performance. We also compare the performance of the proposed solution to the original port knocking implementation.

IV. DESIGN AND IMPLEMENTATION

A. *Design*

To differentiate the proposed solution from the three related port knocking technique, we define the use of OTP S/Key protocol and Diffie Hellman Key Exchange to simplify and reduce the processing time of port knock, but also achieve the intended goal of replay attack prevention.

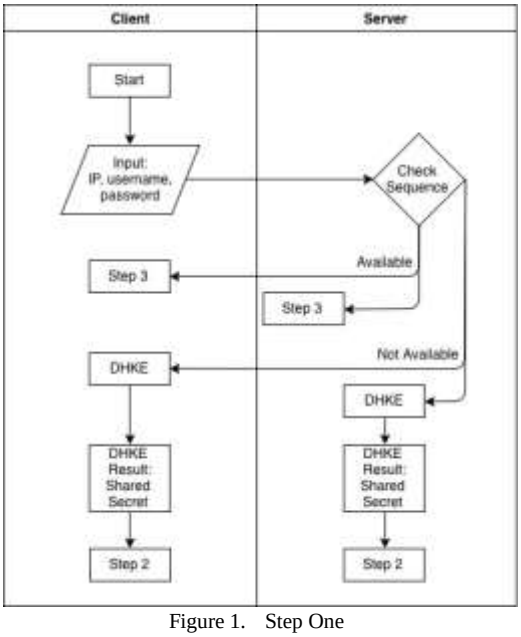

Figure 1. Step One

The proposed solution can be divided into three step. Step one, given in Figure 1, is the DHKE between server and client. Assuming there are no existing OTP sequence in the server and client, the purpose of step one is to generate a shared secret between server and client. If a sequence exists, then step two is skipped and process goes directly to step three.

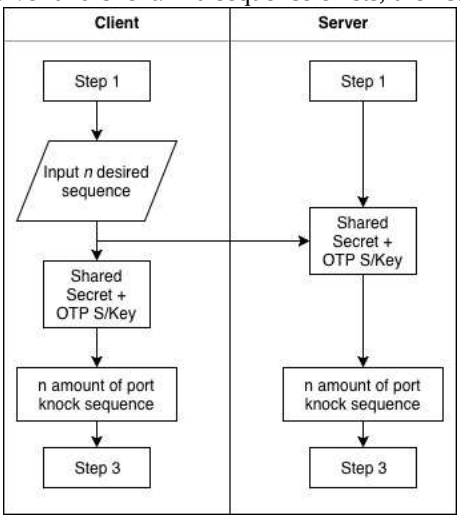

Figure 2. Step Two

After DHKE is done, both client and server have the same shared secret "key" value. This "key" value is then fed to the OTP S/Key to produce the same *n* amount of port knock sequence in the server and client, given in Figure 2. There are no modification on the DHKE protocol, everything is on specification with standard [8].

$$H(key) = h1$$

$$H(h1) = h2$$

$$\ldots$$

$$H(n-1) = hn \qquad (1)$$

The OTP S/Key takes the *n* input from the user, and use it to determine the number of hashes to generate. The hash is produced by applying recursive hash to the shared secret "key" *n* times, storing each hashed value, as in equation (1). OTP S/Key used in this research comply fully to the specification defined [9] [10], with an exception of the hash algorithm, in this research we use the SHA3-256 [11] hash algorithm.

$$h1[0..15] = first\ port$$

$$h1[16..31] = second\ port$$

$$h1[32..47] = third\ port$$

$$h1[48..63] = fourth\ port \qquad (2)$$

After both server and client possess the same list of hashed value, the next process is translating the hashed value into port

ISBN 978-602-53524-0-9

numbers. The hash algorithm used, SHA3-256, has an output of 256 bits, and this needs to be transformed into an integer with the value ranging from 0 to 65,535 (the amount of port numbers in a computer) or 2^{16} bits. To achieve this, not all of 256 bits hash value are used, only the first 64 bits. Hence, we can extract four ports for each hash value, as in equation (2). This transformation process are applied to all *n* amount of hash value.

The process continues with the third step, which is the authentication of client by the server, given in Figure 3.

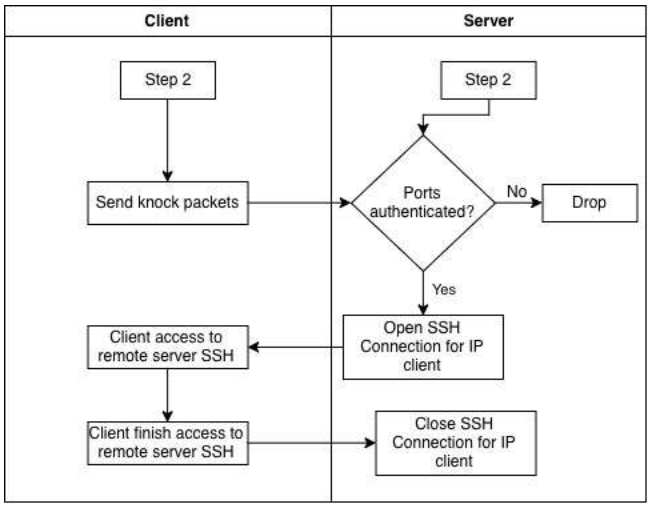

Figure 3. Step Three

B. Implementation

To implement the design, we use Python v.3 [12] in both server and client. In the server, the default configuration is to listen for incoming connection. While the client needs to specify IP destination, username and password for SSH, as in Figure 4.

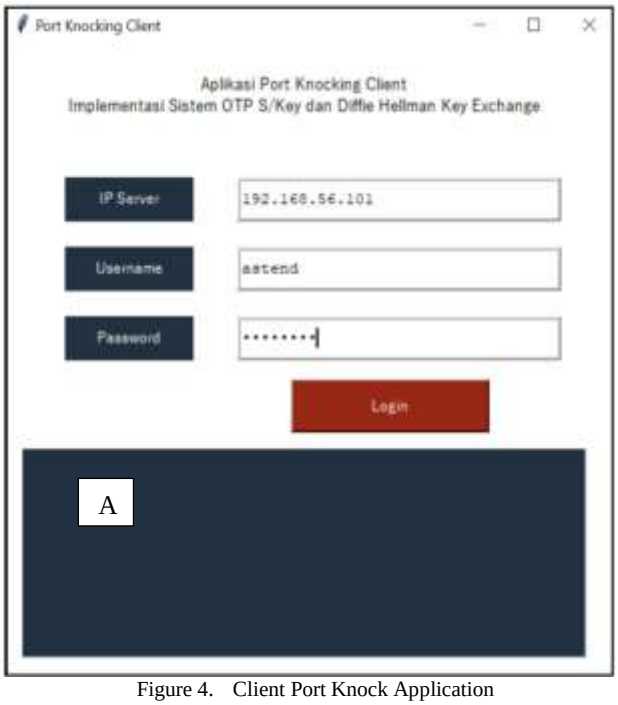

Figure 4. Client Port Knock Application

The connection process can be monitored in the command line interface labelled by the letter "A" in Figure 4. If the authentication is successful, another shell will appear to interact with the SSH session, as in Figure 5.

Figure 5. SSH Session

V. Replay Attack and Performance Test

To demonstrate the outcome of this research, which is to prevent replay attack, the environment are setup as Figure 6, with a network analyzer (Wireshark) tap on R2 Fast Ethernet 0/0.

The original port knocking and the proposed application will use the same topology as Figure 6. To prevent crashing or incompatibility, in any given time, there will only be one type of port knocking application (original or proposed) running on the server and client.

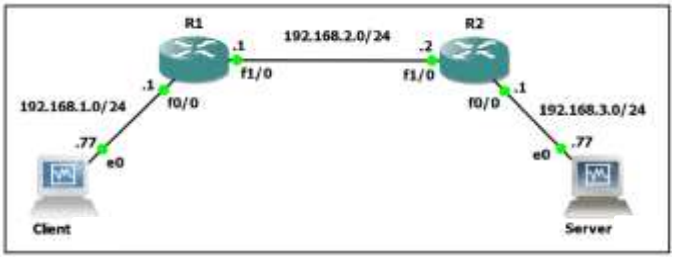

Figure 6. Testing Environment Topology

The result of port usage after five times consecutive open and close is summarized in Table 1.

TABLE I. Port Usage

No.	Open Port Sequence Original	Open Port Sequence Proposed
1	7000, 8000, 9000	55441, 62839, 19876, 53395
2	7000, 8000, 9000	14351, 18315, 4853, 26010
3	7000, 8000, 9000	22729, 19580, 1958, 24421
4	7000, 8000, 9000	40706, 58380, 15860, 1400
5	7000, 8000, 9000	36035, 51660, 62595, 55481

ISBN 978-602-53524-0-9

As shown in Table 1, it is clear when using the original port knock application, the port number and sequence does not change. If an adversary can do the same (network tap/eavesdrop), he/she will be confident enough that the sixth, seventh, and so on port number and sequence in the knock will still be the same. This is the replay attack flaw that exists in the original design of port knocking.

Compared to the original port knock, the proposed solution has port number and sequence changes every time with each successful knock. An adversary will find it hard to correlate between port number sequence on the first knock and any other subsequent knock to guess the sixth port number sequence. If an adversary would attempt a brute force attack on the port, then he or she would have to try $(2^{16})^4$ port sequence combination.

Another aspect to consider is the performance test between the original (Knockd) and proposed solution. This performance test is measured from the first packet sent from supplicant until the SSH port is opened. After ten tests, Table 2 shows the average time consumed by both port knocks.

TABLE II. PERFORMANCE COMPARISON

No	Port Knocks Techniques	Average Time
1	Original (Knockd)	14,4858448
2	Proposed Solution	6,3329799

VI. CONCLUSION

After several comparison tests between the original port knocking technique and the proposed technique, it is clear to conclude the following:

- Combining OTP S/Key protocol and Diffie Hellman Key Exchange in the proposed technique is proven to effectively prevent replay attacks from happening. Furthermore, it's performance is at least twice as fast if compared to the original (Knockd);
- The OTP S/Key protocol provides one time knock port sequence, that although it can still be eavesdropped, but will be worthless for future use;
- The DHKE protocol is essential for setting up the shared secret required by the OTP S/Key protocol to generate the one time knock port sequence list.

REFERENCES

[1] P. Engebretson, The Basics of Hacking and Penetration Testing, Elsevier Inc., 2013.

[2] W. Goralski, The Illustrated Network: How TCP/IP Works in a Modern Network, Elsevier, 2009.

[3] D. Isabel, "Port Knocking: Beyond the Basics," SANS Institute, 2005.

[4] M. Krzywinski, "PORT KNOCKING - Network Authentication Across Closed Ports," in *SysAdmin*, 2003.

[5] F. H. M. Ali, R. Yunos and M. A. Mohamad Alias, "Simple port knocking method: Against TCP replay attack and port scanning," in *International Conference on Cyber Security, Cyber Warfare and Digital Forensic*, 2012.

[6] D. Sel, S. H. Totakura and G. Carle, "SKnock: Port-Knocking for Masses," in *IEEE Symposium on Reliable Distributed Systems*, 2016.

[7] V. Srivastava, A. K. Keshri, A. D. Roy, V. K. Chaurasiya and R. Gupta, "Advanced port knocking authentication scheme with QRC using AES," *International Conference on Emerging Trends in Networks and Computer Communications*, vol. ETNCC2011, pp. 159-163, 2011.

[8] E. Rescorla, "RFC 2631. Diffie-Hellman Key Agreement Method," IETF Network Working Group, 1999.

[9] N. M. Haller, Bellcore and C. Metz, "RFC 2289: A One-Time Password System," Kaman Sciences Corporation, 1998.

[10] N. Haller, "The S/Key One-Time Password System," in *Symposium on Network and Distributed System Security*, 1994.

[11] C. Romine, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, Federal Information Processing Standards Publications.

[12] K. Reitz, Python Guide Documentation, Python Software Foundation, 2018.

AUTHORS PROFILE

Muhammad Yusuf Bambang Setiadji. Lecturer at Sekolah Tinggi Sandi Negara;

Muhamad Arie Taufik Rido Ganatuloh. National Cyber and Crypto Agency Junior Staff.

ISBN 978-602-53524-0-9