An Analysis of Port Knocking and Single Packet Authorization

MSc Thesis

Sebastien Jeanquier
info () securethoughts ! net
GPG Key ID: 0xBE4D6CE8
Supervisor: Dr. Alex Dent

Information Security Group
Royal Holloway College, University of London

September 9, 2006

Contents

List of Figures

Executive Summary

This thesis will analyse the network security concept of Port Knocking and its younger brother Single Packet Authorization and assess their suitability as 'Firewall Authentication' mechanisms for opening network ports or performing certain actions on servers using these mechanisms.

The introduction provides a short history of network security and why this concept has come about at the start of this century. It will also cover the basics of networking and cryptography required to understand the fundamental workings of port knocking systems and the threats and attacks pertinent to them. An overview of both port knocking and single packet authorization and the security aspects involved, including the debated topic of security through obscurity, will enable a clearer understanding of port knocking in actual use and the analysis of implementations of both forms of firewall authentication schemes.

The aim of this thesis is to analyse the security offered by both systems and assess which threats exist in theory and in the real world, and outline the practicalities of using port knocking as part of defence in depth. Finally, this thesis attempts to mention certain possible improvements to port knocking schemes, as well as an overview of alternate uses of port knocking in other aspects of information security.

The two primary implementations that will be analysed are Martin Krzywinski's Port Knocking Perl Prototype and Michael Rash's single packet authorization Firewall Knock Operator (fwknop). In actual use, it was found that the Perl Prototype may be more restrictive due to the long 'knocks' required when encryption is used, and anti-replay features require that state be maintained on both the server and client. The extremely low transmission rate and delivery-order issues involved with port knocking make it less suitable where more data may be required for a secure and practical knock. On the other hand, the single packet authorization implementation, fwknop, uses single UDP packets to transmit authorization data, much in the fashion described in ISO/IEC 9798-2 on entity authentication, but loses the 'knocking' aspect of port knocking, which is a novel and unique delivery mechanism. In its default configuration, fwknop is quite vulnerable to dictionary attacks, simply due to the way in which passphrases are turned into cryptographic keys. A will present a simple tool, fwknop_da, designed to illustrate how a live attacker could intercept fwknop authorization packets and crack them.

Part I

Introduction and Theory

Chapter 1

Introduction

1.1 A Short History of Network Security

An important problem in the domain of network security is the plethora of services running on networked machines, and more specifically, the open ports which allow any user to connect to those services and attempt to attack them in any one of countless ways. In the early years of its inception, around the 1980s, the Internet was designed purely with interoperability in mind. The engineers involved wanted machines to communicate easily, unhindered by the obstacles brought upon by 'unnecessary' security features such as authentication and access control. In fact, security features were, for the most part, unheard of, and programming flaws in network-enabled software coupled with weaknesses in early networks brought about many opportunities for attacks.

So, over the years, we have seen many improvements to protocols and even the network stack itself with the aim of making them more resistant to attack. One primary concern was the notion of authentication, to which there are many solutions, a popular one being the requirement for users to enter a username and password before a service can be used. Many SMTP (email sending) servers now require clients to authenticate themselves by sending some form of username and password before they are allowed to transmit an email message. Previously, SMTP servers were open to anyone who wished to use them. One could simply connect to them using a telnet client, enter a fake FROM address, and send a spoofed email to any recipient. This problem is still highly prevalent today, although on a far smaller scale than we used to see. Security has also been applied at different places in the TCP/IP protocol stack including the invention of IPSec that aims to protect packets at the IP Layer, and SSL/TLS which protects packets at the Transport Layer [60]. Confidentiality, Integrity, and Authentication[1] have become the primary concerns for protocols carrying sensitive traffic. Without these precautions, the information sent over the network could be vulnerable to unauthorised disclosure, modification, or we would simply be unsure as to whom actually sent that information.

In today's world of vulnerability disclosure, exploit releases, worms, and

[1]Traditional information security usually relies on the notions of Confidentiality, Integrity, and *Availability* however the latter does not formally apply to network transmissions.

script kiddies[2], the Internet has become an increasingly hostile environment for businesses and home users alike. Advanced tools which are at anyone's disposal allow attackers to easily discover networked machines, enumerate ports and services running on them, and even whether or not those particular services are vulnerable to a particular exploit [10, 11]. A simple ping sweep and port scan will reveal a large majority of machines and their services, whether or not the scanning machine is authorised to view those resources. Many machines now run some form of Firewall (see Section 2.1.4) to help prevent unauthorised connections to open ports, however, if the machine is needed to run services accessible to the Internet then this is not always an option. There are countless ways to protect information flowing over a network, but if the software running those services has bugs, then protecting that machine against attacks becomes a much bigger problem.

1.2 Enter Port Knocking

One major difficulty in protecting networked machines (especially those running services) is that they are, for the most part, visible and happy to disclose information to anyone who asks. If a hacker[3] finds a corporate FTP server, he can connect to it and it will gladly tell him exactly what version of what FTP software it is running (if the FTP server hasn't been hardened, which is usually still the case). He can then use this information to check whether or not that version of the software is vulnerable to a particular attack which would give him root (admin) access to the server machine (or attempt to brute force the username and password). Many people, unfortunately, do not always have the time to keep all of their machines patched and up-to-date, and even then, many services have so-called 0day[4] exploits for which patches do not even exist yet! So how can they protect themselves against this? One simple answer is to turn off all unnecessary services; another is to use a Firewall to prevent anyone, except a specific group of IPs, from connecting to that service, which is obviously very restrictive in terms of who is able to connect.

How *does* one try to keep a machine hidden from would-be attackers, yet allow legitimate users to connect to services running on that machine? Enter Port Knocking. In broad terms, port knocking is a method for transmitting information across closed ports, with the aim of authenticating users before allowing them, and only them, to access a protected service. The name "Port Knocking" originated with Martin Krzywinski[5] in 2003 [32], and refers to the concept of sending packets to predetermined network ports (see Section 2.1.1), essentially forming what can be compared to as a 'secret knock' on those ports.

[2]In the security field, an unskilled attacker is usually referred to as a 'script kiddie' due to the fact that such attackers mostly scripts that require little-to-no knowledge of security and/or what the attacks actually do. Such attackers are also often quite young.

[3]I apologise in advance to all of the legitimate "hackers" out there, for giving in to the modern-day trend of referring to "crackers" as "hackers".

[4]Zero Day: exploits which are un-released and unknown to the security community or the vendor of the software.

[5]Martin Krzywinski - http://www.portknocking.org

The basic idea was discussed as early as 2001, posted on a German Linux User Group mailing list [7].

This thesis will also cover port knocking's younger brother Single Packet Authorization (SPA), which was developed concurrently by two different groups of researchers. MadHat Unspecific and Simple Nomad are one team of researchers who presented their ideas at BlackHat 2005 [37]. One popular implementation of SPA which has been gaining interest in the security community is fwknop, developed by independent researcher Michael Rash [46], and will be the main SPA implementation covered in this thesis. Both port knocking and SPA have the same goal, however the methods they employ are significantly different.

In this thesis, both standard port knocking techniques as well as single packet authorization will be referred to as 'Port Knocking' for simplicity, as all implementations are essentially 'firewall authentication mechanisms' and aim to perform the same basic task of user authentication at the networking level.

Chapter 2

Technical Theory

2.1 Networking

Port Knocking schemes are, for the most part, network security methods for authenticating users and authorizing them to use a specific service (this may include performing an action on the protected machine). For these reasons it will be important to grasp the basics in host-based networking in order to understand how port knocking schemes function. This chapter will serve as a very basic introduction to the aspects of networking which the author feels are most relevant to achieving this understanding.

2.1.1 Ports

Those unaccustomed to host-based networking sometimes have trouble coming to terms with the notion of a 'port' on a computer. In the simplest of terms a port is a virtual door (represented by a 16-bit integer) which allows the computer to keep track of which pieces of data are destined for which application or service. A computer has 65535 of these ports [60]. Networking (transport layer) protocols such as TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) both use the concept of a port when transmitting packets to and from networked hosts. Some other protocols such as ICMP, however, do not use ports when transmitting information.

The port number (when used) is included in networking packets and is interpreted not only by sending and receiving hosts, but also by intermediate routers and firewalls. A firewall can be configured to allow or deny certain packets based on their destination port. When a service is listening for requests on a port, that port is said to be open, and clients can connect to the service. If no service is listening, then the port is considered closed. A client cannot connect to a closed port.

Many ports, especially those within the 0-1023 range, are reserved for use with specific services. The vast majority of web servers, for example, run on port 80. There exist many services with their own port numbers such as File Transfer Protocol (port 21), SSH (port 22) and the Post Office Protocol (port 110). Although these port numbers are 'reserved' for those services, it is still possible to run a web server, for example, on port 22, if the administrator felt like doing so. The Internet Assigned Numbers Authority (IANA) is responsible for assigning TCP and UDP port numbers to specific services, and their list of officially assigned port numbers is regularly updated [26].

2.1.2 TCP, UDP and ICMP

TCP, UDP and ICMP are three of the most important networking protocols used regularly in modern networks. Transmission Control Protocol (TCP) is a stateful protocol that allows the two machines to create a connection between themselves and exchange information. A connection can be defined as two machines that have mutually 'agreed' to communicate. Such a connection is established by performing the 'Three-Way Handshake' (see Section 2.1.3). This can be compared to making a telephone call: the initiator dials the number, the receiver picks up and says "Hello?", at which point the initiator also says "Hello!", and the conversation can begin until it is ended by either end. Most applications, such as Email, Web Browsing, and File Transfers, use TCP connections to transfer information.

The opposite of TCP is the User Datagram Protocol (UDP) which is stateless and so no formal connection is established between communicating hosts. This can be compared to a postal letter: the sender writes a letter and sends it off to its destination. The letter might arrive at its destination, or it might not, either way the sender will not receive any confirmation. A host sending UDP packets to another host will receive no acknowledgement as to whether or not the packets have been received, this makes UDP a lot faster than TCP, although far less reliable. UDP is suitable for applications which require a rapid rate of transmission, and where reliability is not of up-most importance, such as Audio/Video Chat.

The Internet Control Message Protocol (ICMP) is one of the core protocols of the Internet protocol suite, and is used by networked computers to send error messages, for example when a specified port cannot be reached (see Section 3.1). In the case that a service is not available or a host cannot be reached, there are a number of 'control messages' that end hosts or intermediate routers can use in order to inform other devices of these errors. One example is the ICMP_PORT_UNREACHABLE (ICMP Type 3, Code 3) which informs a requesting host that the requested port cannot be reached for some reason [27].

Applications do not tend to use the ICMP protocol directly (except for the ping command), but in certain cases ICMP can be used to transmit small amounts of information within the *data* field of the ICMP packet.

2.1.3 Three-Way Handshake

The Three-Way Handshake is the protocol that computers use in order to establish a TCP connection with each other.

1. The initiating machine will send a 'Hello' (formally called a SYN) packet to a specified host on a specified port. For example, if you browse to

http://www.foo.com, your computer will first send the server a SYN packet on port 80 (the default web server port) to the server at www.foo.com. If port 80 is not open on the web server, then your client will not receive a reply (and the connection will fail).

2. However, if port 80 is open, then the web server is listening and will reply to the client with a SYN/ACK packet, acknowledging that it received the first (SYN) packet and requesting a confirmation to complete the connection.

3. Finally, the client will send back an ACK packet, signalling that it confirms the connection (see Figure 2.1).

At this point, both computers keep track that they are connected to each other. It is also important to note that some machines will only log a connection[1] once the full Three-Way Handshake has been performed.

The connection is ended by one side or the other, by sending a FIN packet to the other end, who then replies with a FIN&ACK packet, and finally the initiating side sends back a final ACK packet.

2.1.4 Firewalls

Firewalls are an essential network component when it comes to controlling the flow of access in networked environments. In short, the goal of a firewall is to allow controlled connectivity between areas of differing trust levels, through the enforcement of a security policy and connectivity model, based on the principle of least privilege[2] [60]. Firewalls can be either hardware, which sits on a network between a trusted side and an untrusted side; or software, which runs on the hosts themselves[3]. In both cases the purpose of the firewall is to provide a logical barrier to prevent unauthorized or unwanted communications between different areas of a computer network.

The 'Access Control' mechanisms offered by a firewall rely on a set of administrator-defined rules, which are then applied to each and every packet flowing through the firewall. The default rule, which helps satisfy the principle of least privilege, is the 'default deny' rule where all traffic is rejected unless explicitly allowed. In this manner, one can be sure that no access is allowed except for the rules that specify an 'allow' condition. In most firewall packages there are two distinct ways that a firewall can deny traffic [55]:

- **Reject/Deny:** Different firewalls refer to this rule as Reject or Deny although they both perform the same action. Using the REJECT/DENY

[1]Logging a connection can be defined as: writing the details of the connection (source, destination, port, etc) to a log file of some sort, usually used to keep track of which hosts have connected to the local machine.

[2]"The principle of least privilege states that a subject should be given only those privileges that it needs in order to complete its task. If the subject does not need an access right, then the subject should not have that access right." [6]

[3]IPTables/netfilter (http://www.netfilter.org) is one of the more widely used packet filtering firewalls on Linux machines, and ipfw is the firewall on BSD-based Unix.

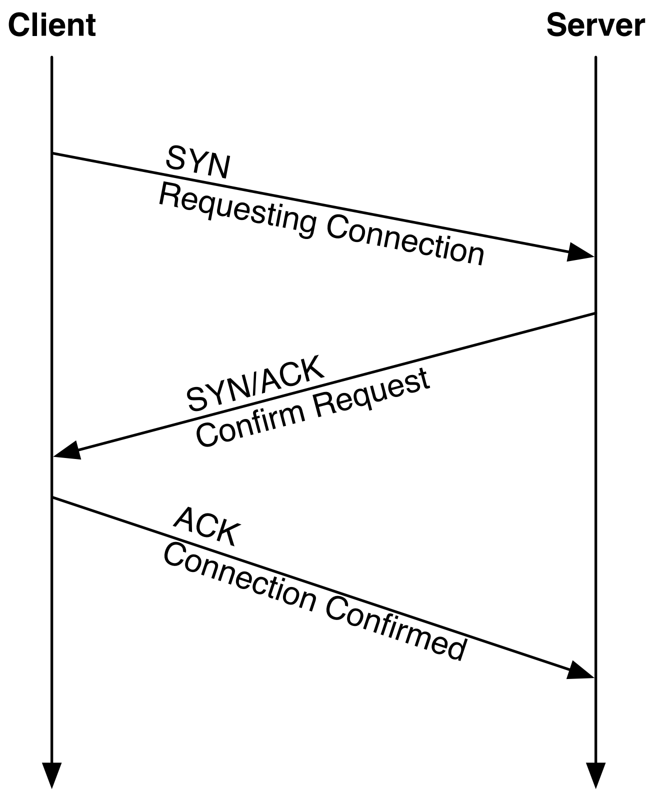

Figure 2.1: Three-Way Handshake

rule will instruct the firewall to drop packets[4] and send an ICMP_PORT_UNREACHABLE back to the initiator. The connection will be rejected, but the initiator will know that a host exists at that IP, and is denying access.

- **Drop:** Using the DROP rule instructs the firewall to be more silent in its denial. Packets that arrive are dropped *without* sending an ICMP_PORT_UNREACHABLE back to the initiator. The connection will be rejected, but the initiator will simply assume that no service is running on the target host (or that the target host does not exist).

The distinction between these two rules will become clearer as the port knocking mechanisms are explained.

As mentioned above, firewalls are purely access control mechanisms – with a fundamental lack of authentication. A firewall will simply compare an incoming packet to see whether or not its requested access is allowed by the firewall rules. There is no mechanism to allow or deny access based on some form of strong authentication of the user requesting access. Due to this, firewall configuration is either extremely restrictive (eg. access restricted to certain IP addresses[5]), or unrestrictive (eg. anyone can access the FTP port). In such a way, it is difficult to configure a firewall to protect a host which must be accessible to clients whose IPs are not known prior to them connecting.

2.2 Cryptography and One-Way Hash Functions

Cryptography (from Greek *kryptós*, "hidden", and *gráphein*, "to write"), the transformation of data into a form unreadable by anyone without a secret decryption key, is said to be about *"communication in the presence of adversaries"* [51].

Cryptography has the ability to provide a number of services which aid us in protecting our information in various ways as it is sent across networks or stored on physical media. Confidentiality is a crucial element of network communications when private information is being stored or transmitted. The use of encryption has allowed us to protect such information and prevent it being disclosed to unauthorised parties. Similarly, data integrity ensures that our information is not modified in transit, and that we can trust that the information received is as-intended. The slightly more contemporary field of public key cryptography has the added ability of providing non-repudiation whereby it can be proven that an individual did indeed send a particular piece of information. Cryptography has proven to be extremely important in authentication protocols, as it is necessary for certain pieces of information to be protected from unauthorised modification in order to result in successful authentication.

Cryptographic algorithms can be divided into two main categories, Symmetric (Secret Key) Cryptography, and Asymmetric (Public Key) Cryptography.

[4]Dropping a packet simply means to discard/ignore it, as it will not be needed.

[5]An IP address is a unique number that devices use in order to communicate over a network. Each host has its own IP address, much like each house has its own postal address.

Each category has its own unique set of features and abilities which may apply to different situations.

2.2.1 Symmetric Cryptography

Symmetric cryptography refers to algorithms which use the same key to encrypt and decrypt data (see Figure 2.2). These keys usually represent a 'shared secret' between all users who may need to communicate using the same algorithm. An example of a symmetric algorithm is the Data Encryption Standard (DES) which was selected as an official Federal Information Processing Standard (FIPS) for the United States in 1976 [43]. DES was replaced by the Advanced Encryption Standard (AES) in 2001 [44]. Both DES and AES are known as block ciphers. Block ciphers are algorithms which operate on a groups of n-bits of data called blocks, where n (the size of the block) is dependent on the algorithm being used. DES encrypts data in blocks of 64-bits using a keylength of 56-bits, whereas AES encrypts data in blocks of 128-bits and supports keylengths of 128, 192 or 256-bits [54]. Symmetric block ciphers make use of different *modes of operation* when encrypting data. These modes of operation allow for messages longer than the encryption algorithm's block length to be encrypted. Cipher Block Chaining (CBC) mode, for example, is a popular mode of operation that helps increase the security of a block cipher. It works by ensuring that each each ciphertext block depends on the values of all previous message blocks. Each new ciphertext block relies on the current plaintext block and the previous ciphertext block [54]. Thanks to this mechanism, if two identical plaintext blocks exist within a given message, they will both produce different ciphertext blocks, which makes it much more difficult to notice a particular plaintext block from its ciphertext.

A Message Authentication Code (MAC) is a short piece of information, similar to a hash code (see Section 2.2.3), which provides both origin authentication and integrity protection of a message. MACs are generated by block ciphers and use symmetric secret keys to ensure that only those who know the key can modify, or verify the message. MACs on their own, however, do not provide any confidentiality. As the data must be sent together along with its corresponding MAC. A popular MAC algorithm is the CBC-MAC based on the CBC mode of operation.

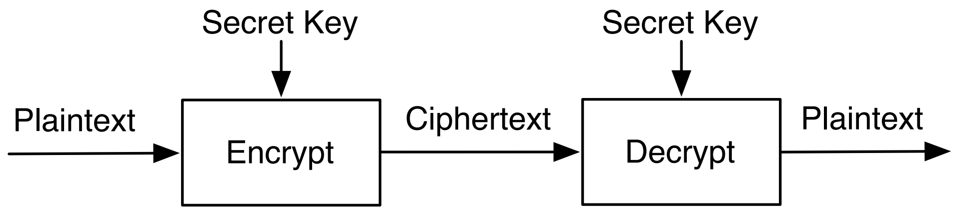

Figure 2.2: Symmetric Cryptography Encryption and Decryption

Due to the fact that a group of users communicating securely, using symmetric cryptography, must share the same key, this creates certain key management issues when a single user leaves the group. A new key must then be generated and distributed to the remaining group members in order for further communications to remain confidential.

2.2.2 Asymmetric Cryptography

Asymmetric cryptography, also known as Public Key Cryptography (PKC), refers to the fact that different keys are used in the encryption and decryption processes. These keys are mathematically related, but the private key cannot be derived from the public key. In public key cryptography, the private key is kept secret, whilst the public key can be distributed freely. In such a way, anyone can use a user's public key in order to encrypt a message to them, but only the user holding the corresponding private key can decrypt that message [54]. This can be compared to the notion of giving someone a padlock for them to send you private information. They can use the padlock to secure the container, but only the padlock's owner (and key holder) can unlock it and recover the contents of the container. The key cannot be re-created by examining the lock.

One of the popular public key algorithm is called RSA, developed by Ron Rivest, Adi Shamir, and Len Adleman at MIT in 1978 [52]. The RSA algorithm relies on the inherent difficulty involved in factoring the product of two large prime numbers. Another popular algorithm is ElGamal, which relies on the difficulty of calculating discrete logarithms [20]. Due to their mathematical nature, most public key algorithms are very slow compared to their symmetric counterparts. Pretty Good Privacy (PGP), originally designed by Phil Zimmermann in 1991, is a hybrid cryptosystem, whereby it employs both symmetric and asymmetric techniques. PGP also provides a means for binding public keys to users' identities. In general use, PGP operates as a 'web of trust' where users sign each others' keys in order to show that their keys are trusted by other users, although PGP now supports the use of Public Key Infrastructures [54].

Public key cryptography introduces some interesting key management issues when a user wants to verify that a public key actually belongs to a given person, and not an attacker. Due to this, many environments using widespread public key cryptography must set up a Public Key Infrastructure (PKI), where a Trusted Third Party (TTP) is set up to vouch for users' identities and digitally 'sign' people's public keys, enabling other users to verify the identity of a public key's owner.

Digital Signatures

Digital Signature schemes are cryptographic schemes derived from public key cryptography, which allow a recipient to verify the origin and integrity of a message. In such schemes, each user has their own Signing Key and Verification Key. In this case the verification key can be freely distributed to those who will need to verify signatures created by the signer. A digital signature is produced by inputting the message and secret signing key into a signature algorithm. The signature is then sent along with the message to the recipient. To verify the signature, the recipient inputs the message and the signature into a verification algorithm which outputs a 'yes/no' to indicate whether or not the signature on this message is valid. If the signature is valid it provides the verifier with confirmation that the message did indeed come from a given sender, and that the message was not altered in transit [39] (see Figure 2.3).

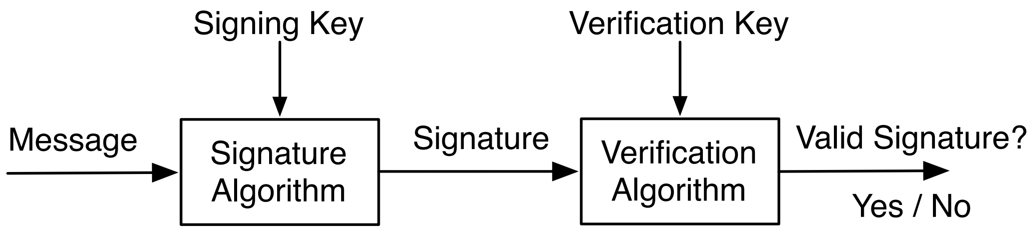

Figure 2.3: Signing and Verifying a Message using Digital Signatures

Bruce Schneier describes the aim of signatures as follows [54]:

1. The signature is authentic. The signature convinces the document's recipient that the signed deliberately signed the document.

2. The signature is unforgeable. The signature is proof that the signed, and no one else, deliberately signed the document.

3. The signature is not reusable. The signature is part of the document; an unscrupulous person cannot move the signature to a different document.

4. The signed document is unalterable. After the document is signed, it cannot be altered.

5. The signature cannot be repudiated. The signer cannot later claim that they did not sign it.

Digital signatures bind the identity of a person to a document, whilst ensuring that any modifications to that document would be easily detectable. The non-repudiation aspect of digital signatures is one of the features of asymmetric cryptography which make it so appealing.

2.2.3 Cryptographic One-Way Hash Functions

Hash functions are algorithms which take an arbitrary-length input, and produce a fixed-length output, often called a message digest or a hash. Cryptographic One-Way Hash Functions are expected to have the following attributes [54]:

1. **Pre-image Resistance**: Given a hash h, it should be hard to find any other message m such that $h = hash(m)$.
2. **Second Pre-image Resistance**: Given an input m_1, it should be hard to find another input m_2 such that $hash(m_1) = hash(m_2)$.
3. **Collision Resistance**: It should be hard to find two messages m_1 and m_2, such that $hash(m_1) = hash(m_2)$.

By including a ‘secret’ value and some form of timestamp together with the input, the resulting hash is unique to that time and to the group of people who know the secret value (often a password). The notion of a timestamp will prove to be extremely important, as it will allow the recipient to check exactly when the hash code was produced. It will also make it a lot more difficult for an attacker to capture the hash code and successfully replay it at a later time.

MD5 and SHA

Two widely used hashing algorithms are MD5 (Message Digest Algorithm 5), designed by Ron Rivest in 1991 [53], and the SHA (Secure Hash Algorithm) family of hash algorithms, designed by the National Security Agency (NSA) and published as a US government FIPS standard in 1993 [41]. MD5 creates a 128-bit hash, whereas SHA-1 creates a 160-bit hash [54]. Below is an example of MD5 in use:

MD5 (“Hello World!”) = ed076287532e86365e841e92bfc50d8c
MD5 (“Hello World”) = b10a8db164e0754105b7a99be72e3fe5

An important feature of good hash functions is that a small change in the input should produce a large change in the output. In the example above, the two inputs only differ by one character, however, the resulting hashes are completely different. Although both MD5 and SHA-1 have been ‘broken’ [31, 63] in the past couple years, their use is still widespread as the chance of an attacker orchestrating a successful attack based on a collision is still considered to be quite low.

SHA-1 is the hash algorithm of choice for various security applications and protocols such as TLS, SSL, SSH, S/MIME, IPSec and PGP. Many cryptographers have recommended that the SHA-2[6] group of algorithms be adopted to replace MD5 and SHA-1 where possible, as these algorithms produce much longer hashes, thus significantly reducing the chance of finding collisions.

[6]The SHA-2 group of algorithms produce 224, 256, 384 or 512-bit hashes [42].

Part II

Port Knocking 101

Chapter 3

Basic Port Knocking

The concept of port knocking has been the topic of heated debates in the security community, with large numbers of posts being posted on technology and security site Slashdot.org [40, 34, 56, 57, 58, 59, 23]. Interestingly enough the benefits of port knocking as a security mechanism, although contested, have never been ruled out. In other words, the security community has yet to find a good place for port knocking, but at the same time cannot reach the conclusion that port knocking is insecure or useless. The ongoing discussions about port knocking tend to revolve about *what* it is exactly that the mechanism tries to achieve.

When it comes down to it, port knocking's primary aim is to provide an extra layer of protection through the use of authentication with the added benefit of concealment. As described in Section 1.1, networks were originally designed to work seamlessly, and thus we find ourselves relying on a structure which offers no authentication before one machine can connect to another. Port Knocking attempts to fill the void by adding (not replacing) a layer of security, which can be used to authenticate users before they are given access. From a security standpoint it should be blindingly obvious that there is no good reason for the average unskilled attacker to be able to detect SSH running on port 22 of a server. If only authorized users are allowed to use the SSH service, then it should make sense that only authorized users should be allowed to connect to it in the first place.

Port Knocking allows administrators to keep the (potentially vulnerable) service hidden from the public, whilst making it available to authorized users, without the risk of making the machine vulnerable should the port knocking mechanism fail. More on this later, see Chapter 4.

3.1 Vanilla Port Knocking

In order to analyse port knocking as a security mechanism, it is essential to outline its basic workings. The crucial element of port knocking relies on a completely closed firewall which is set to DROP all packets that arrive. This means that all received packets should be dropped and no response sent out (as opposed to the REJECT/DENY state which drops packets and sends a

ICMP_PORT_UNREACHABLE back to the client). The significant difference is that one gives away the presence of a listening host, whereas the other remains absolutely silent and so makes it impossible to determine whether or not a machine exists at that address (unless sniffing - see Section 4.2.2). At this point we have a completely silent, inaccessible machine. This is relatively secure, yet also relatively useless.

The next step is to find a way to connect to the server or perform some kind of action on it. A port knocking daemon sits on the server and watches packets as they are dropped, waiting for a sequence of packets arriving at a predetermined set of ports in order. The client who wishes to connect to the server sends SYN packets (the first packet in a TCP connection) to the predetermined ports in order, for example: ports 100, 110, 120, 130. The server will receive these packets and drop them silently, however, the port knocking daemon will see these incoming packets and recognise the valid 'knock' on the ports. Once the proper ports have been knocked on, the daemon can execute any predetermined action, for example: open port 22 (SSH) for the IP that knocked (see Figure 3.1). The user who knocked can then connect to the protected machine, and if set up properly, can knock on ports 200, 190, 180, 170 in order to close port 22. Most good implementations, however, simply close the port automatically after a given amount of time has elapsed.

The significant features we can see so far are:

1. **Concealment:** the server's firewall is set to DROP all packets and so a scanning or probing attacker will have no clues as to whether or not the server exists, let alone what services it is running.

2. **Service Protection:** services running on the host (such as SSH) are protected from attacks on unpatched vulnerabilities. This feature serves as a useful stop-gap and gives administrators time to patch their vulnerable systems. One important aspect of this port knocking feature is the fact that it can actually protect against 0day attacks, which are considered extremely difficult to defend against. Even if an attacker possesses an exploit for a vulnerable version of a service running on the protected server, they have no way to mount an attack and deliver that exploit, as all ports on the server are closed to the attacker.

3. **User Authentication:** by watching the incoming traffic for a predetermined knock, which essentially acts as a secret key known only to the users trusted to use the protected server, the port knocking daemon is able to authenticate the user at the other end *before* allowing them to connect to any (potentially vulnerable) services.

Even without going into much depth, it is easy to see that this basic system is fundamentally flawed. Anyone able to monitor the network traffic between the Client and the Server will be able to pick up the proper knock sequence and use it for themselves. Although such a system is fine for protecting the server from the average script kiddie looking for open ports, a more determined

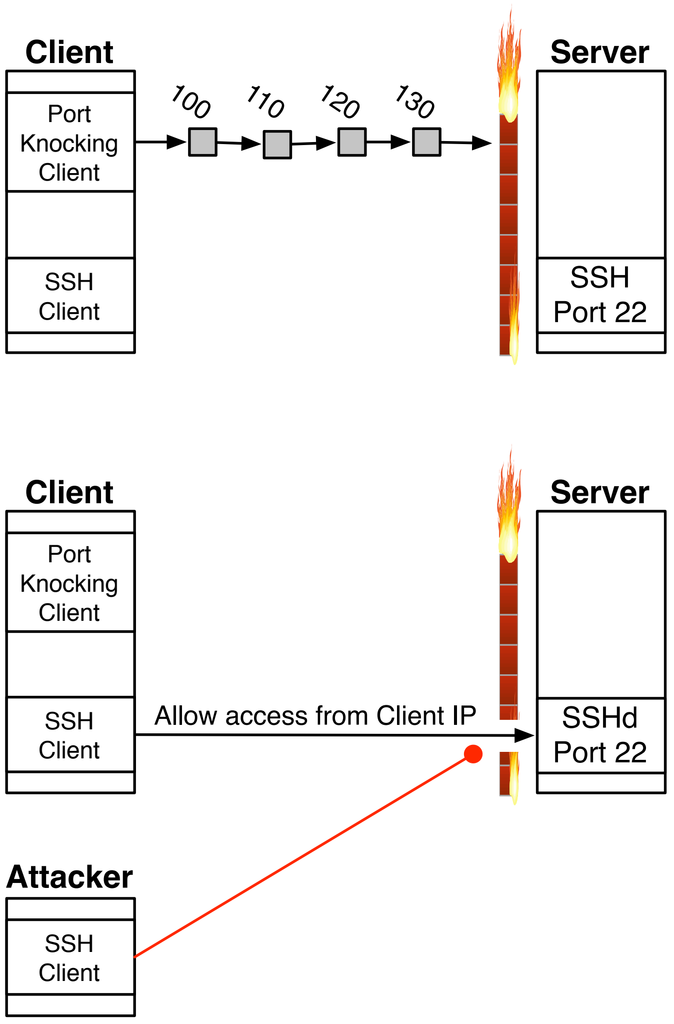

Figure 3.1: Using Port Knocking to Open Port 22

hacker has numerous options at his disposal should he choose to attack this system. An extremely basic 'implementation' by Daniel De Graaf [16] is possible using Linux's built-in firewall, IPtables, simply by adding a few lines to the configuration file, requiring a client to 'knock' on one or more pre-determined ports before access to port 22 will be granted. A more advanced implementation of port knocking, by one of the pioneers of the concept, Martin Krzywinski [33], will be covered in detail in Section 6.1.

3.2 Vanilla Single Packet Authorzation

Single Packet Authorization (SPA) and port knocking have the same aim but significantly different delivery mechanisms. In SPA the knock, which is called an Authorization Packet (AP), is encoded within a single packet. This can provide certain advantages not found in traditional port knocking schemes, such as eliminating the problem of out-of-order packet delivery. In traditional port knocking, if a knock sequence arrives out of order, which can easily happen as the server is not sending back any acknowledgements, then the port knocking daemon will not recognise the knock and thus access will not be allowed (see Section 5.1). SPA simplifies the process by encoding all of the necessary information into a single packet, typically UDP or ICMP. The information encoded in these packets can be as simple as a timestamp (for replay protection), client IP address, and password combination, for example:

- **Timestamp:** 200608062127 (21:27 6th August 2006)
- **Client IP:** 192.168.1.100[1]
- **Password:** secretpassword

In this very simple example, the server would be configured for a single user and would execute a single command, for example to open SSH port 22 for the client IP for 5 minutes. The server would also be set up to keep a record of the last valid authorization packet it received, in order to prevent an attacker from replaying an old AP (see Section 4.2.2). The resulting AP that is sent to the server is usually constructed by feeding the fields above into a hash function (see Section 2.2.3). In this case the resulting hash, using MD5, would be:

MD5 ("200608062127:192.168.1.100:secretpassword") = 9c6f2af8e1a0f841467f0a1de39f53c8

This hash would then be packed into a UDP packet and sent off to the server. Upon receiving the packet, the SPA daemon would recalculate the hash by hashing the password (which it knows), the current date and time (accurate in minutes), and the IP address of the client that knocked (which can be found in the IP header of the UDP packet [60]). If the resulting hash is the same as the

[1]For the purposes of this example we are assuming that both the client and the server are on the same local network. A private IP address can create problems for port knocking schemes due to Network Address Translation (NAT). See Section 5.1.

one received, then port 22 would be opened for the client IP. If the hashes do not match, or the received hash had already been received previously (ie. the hash is a replay of an old hash), then no action is performed. An extremely basic implementation of SPA, that offers no replay protection, is ‘coarseknocking’ by Andre Luiz Rodrigues Ferreira [21]. A more advanced implementation of SPA, Firewall Knock Operator (fwknop) by Michael Rash [46], will be covered in detail in Section 6.2.

3.3 Threats

Before we delve into more detail about the threats against port knocking, it is essential to enumerate our key players.

- **Server:** this machine is the one protected by port knocking and runs a completely closed Firewall, and a port knocking or SPA daemon capable of executing commands on the server (such as manipulate firewall rules).
- **Client:** this machine will be attempting to connect to the server by providing the correct knock sequence (or authorization packet).
- **Eve (attacker):** is a passive eavesdropper. She has the ability to observe all of the interactions between the Client and the Server but can not modify data in transit between the two.
- **Mallory (attacker):** is an active attacker. He has the ability to sit between the Client and the Server on the network and watch and manipulate the traffic flowing between them.
- **Trudy (attacker):** this attacker has no access to any protocol exchanges between the Client and the Server, and thus has no ability to observe nor modify to those exchanges. Trudy must attempt to compromise the Server by other means. This is the most common type of attacker.
- **Trent:** this machine is a Trusted Third Party (TTP) which can be used to send traffic from the Client to the Server via another route, thus bypassing Eve and/or Mallory. **Note**: this is not always possible depending on the position of Eve and/or Mallory on the network!

Port knocking, as with any network authentication protocol, is potentially vulnerable to various kinds of attacks from a number of different vectors. As illustrated above, is it important to design an authentication mechanism with the assumption that an attacker could *potentially* place himself anywhere on the network. Most attackers will not have access to any network traffic, such as Trudy, and will have to resort to simpler methods such as brute force in order to bypass strong authentication mechanisms. It is essential, however, to assume that all attackers have the knowledge and capabilities of a user such as Mallory, with the ability to watch and modify all network traffic between the client and the server.

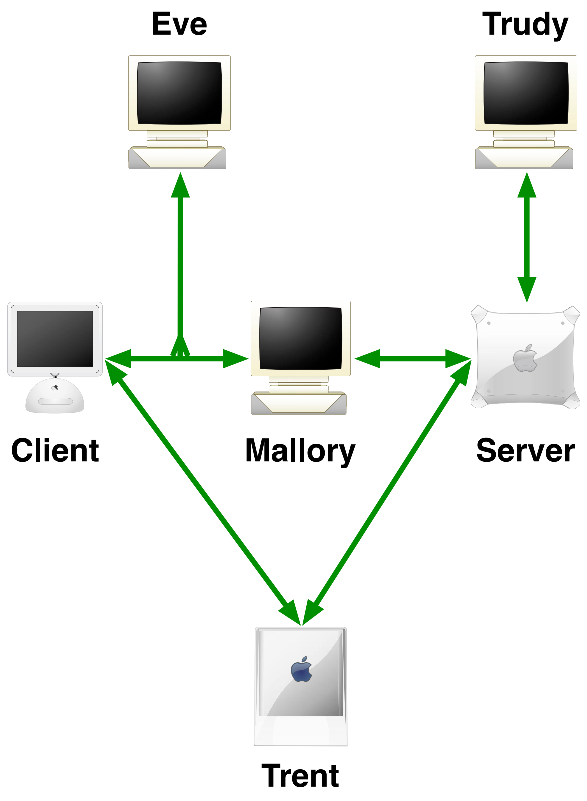

Figure 3.2: Network-view of threats against Port Knocking

Chapter 4

Port Knocking - A Firewall Authentication Scheme

Now that the basic forms of port knocking have been outlined, it is possible to assess such schemes in terms of their viability as network security mechanisms. There exists a very important concept in the field of information security which is commonly referred to as 'Defence in Depth'. This refers to the act of securing one's resources without solely relying on a single protection mechanism. Sometimes this relies on the use of redundancy in the security mechanisms which are put in place, or several mechanisms may be used to protect a particular resource. In plain English it can be compared to not putting all of your eggs in one basket.

Security, and more precisely in this case 'network security', can be thought of as an onion - where each layer of the onion provides an extra layer of protection. It is essential to understand what layers exist in any given security system and what purpose they serve. Many people question the viability of port knocking for a number of reasons. Some assume that it is an authentication layer that is supposed to replace the authentication of the services running below it. Others coin the term 'Security through Obscurity' when they discredit port knocking, which shows that either they misunderstand security through obscurity, or they misunderstand the concept of port knocking.

As this seems to be a highly misunderstood area surrounding this topic, I will aim to clarify why port knocking and SPA are neither security through obscurity, nor redundant authentication mechanisms.

Peeling the Onion

Most current-day network services require some form of user authentication before a connection can be made. Services such as SSH[1] and FTP both require a username and password when a connection is opened. Alternatively, SSH has the option of only allowing cryptographic keys to be used in the authentication process, denying the use of username and password combinations.

[1]SSH will be used in these examples as it is a service which allows full remote access to the machine on which it is running. This is often a desirable service for system administrators, yet is open to the same vulnerabilities as any other networked service.

So in essence, SSH and FTP are *layers* which exist to protect the system and authenticate users before some kind of access is granted. The problem with networked services is that they are potentially vulnerable to attack, for example, by exploiting bugs in their code, or to simple dictionary attacks[2].

For these reasons, port knocking allows the services to be hidden from the world, until a valid authenticated user attempts to connect. Figure 4.1 below depicts how a system running SSH could be protected using a port knocking layer. Remember that in this configuration, the firewall (see Section 2.1.4) is set to drop all traffic silently until a port is opened by the port knocking daemon.

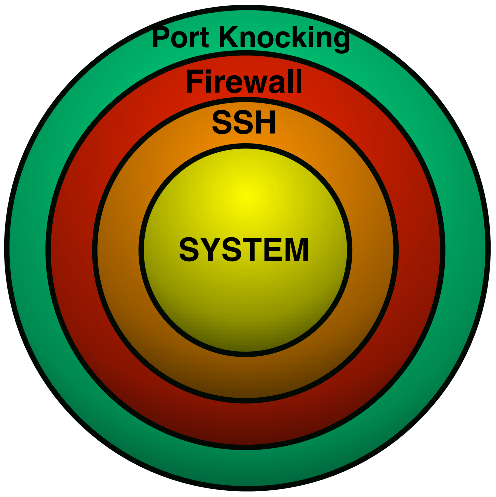

Figure 4.1: Port Knocking as an additional layer in the Defence in Depth ‘onion’.

One major concern about port knocking is the question of what would happen if the mechanism failed. This turns out to be quite a simple scenario to assess, due to the fact that port knocking operates with a completely closed firewall. Figure 4.2 depicts a situation where the port knocking daemon fails ‘safely’. The firewall remains closed, meaning that nobody can connect to any services. Admittedly this results in a classic Denial of Service (DoS), but the system remains secure which may be the preferred result in certain environments.

The other way that port knocking can fail, depicted in Figure 4.3, is the ‘fail open’. This situation can happen, for example, if a user knocks to open a port, and the port knocking daemon fails and thus the port is not closed automatically as it should be. Here the firewall is left in an open state[3], thus

[2]A Dictionary attack is where the attacker uses a dictionary of words and attempts to login to a specified username, trying each word in the dictionary as the password, one by one. Dictionary attacks are easier to detect and block, whereas unknown exploits are much more difficult to defend against.

[3]Note that an ‘open state’ usually means that only some ports are left open (eg. only port 22 open), and not necessarily “all ports open”. The firewall could be all-closed except for one port.

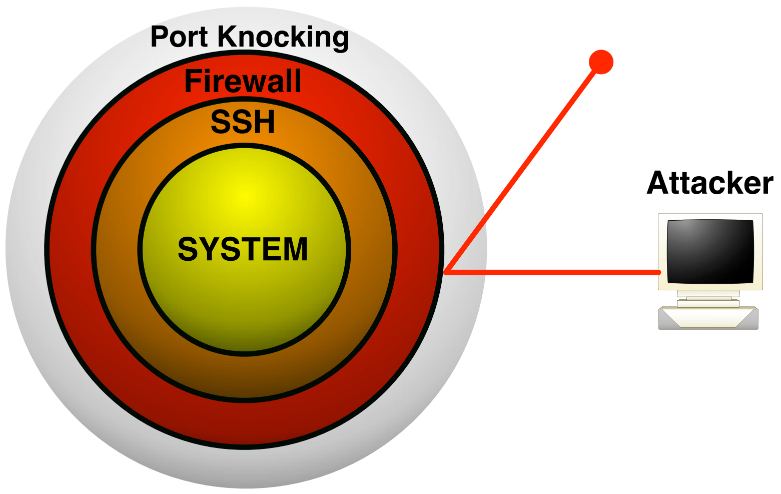

Figure 4.2: Port Knocking Fails Safely - all ports are closed.

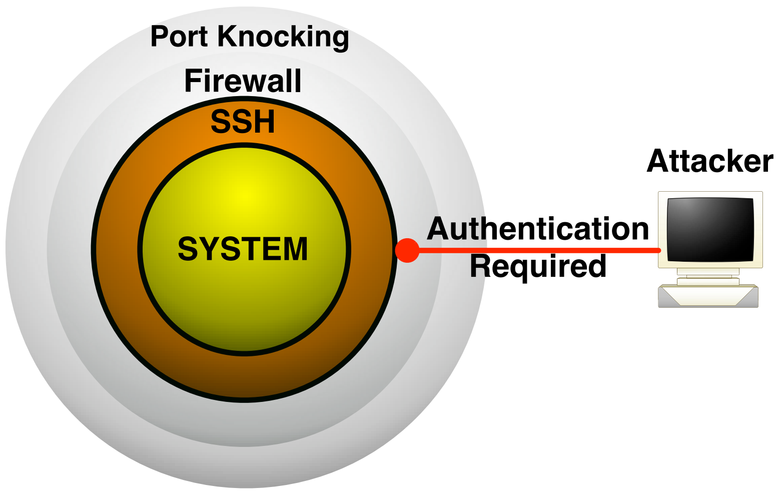

Figure 4.3: Port Knocking Fails Open - attacker must authenticate to underlying service.

allowing connections to the SSH daemon on port 22. This state may result in a DoS for any additional users who wish to connect, and in most cases does not result in an overly open system, as most port knocking implementations only open the port to the IP of the client that knocked. However, a less complex implementation could open ports fully (ie. to any IP), and although this means that the port knocking mechanism has been circumvented, from an attacker's perspective, he would still have to either authenticate to the underlying service or attempt to exploit it - the attacker is essentially back to square one, just as he would have been had port knocking not been employed in the first place! More importantly, compromise of the port knocking mechanism does not result in the penetration of the underlying system [34]!

4.1 Security through Obscurity

"Security through Obscurity" is when the details about a security implementation or mechanism are hidden, in order to protect the mechanism from being analysed and flaws discovered, which may enable the mechanism to be compromised in some manner.

This quote from the preface of *Applied Cryptography* by Bruce Schneier [54] presents a good example of how 'obscurity' is used in attempts to increase security:

> "If I take a letter, lock it in a safe, hide the safe somewhere in New York, then tell you to read the letter, that's not security. Thats obscurity. On the other hand, if I take a letter and lock it in a safe, and then give you the safe along with the design specifications of the safe and a hundred identical safes with their combinations so that you and the worlds best safecrackers can study the locking mechanism – and you still can't open the safe and read the letter – thats security."

Looking again at the concept of port knocking, and indeed all of the multitudes of implementations that exist, it should be quite clear that there is nothing inherently *obscure* about it. The concept of port knocking is clearly described, in all its forms, and practically every implementation available is open source and available for peer review. The confusion arrives due to the fact that one of port knocking's *aims* is to hide a server from unauthenticated access requests. That's not obscurity. . . that's *concealment.*

Concealment in itself is a benefit, but port knocking schemes do not rely on the assumption that the protected host *must* stay concealed! It makes no difference if, by some means, an attacker should discover that port knocking is being used to control the opening and closing of ports. In Section 4.2, attacks are outlined in which an attacker does exactly this, however in no way does this decrease the effectiveness of port knocking as a whole.

This author has tested an analysed several implementations of port knocking, nearly all of which involve concealing the host by means of closed ports (see Section 3.1). Due to the open source nature of these implementations it was possible to determine which implementations offer what levels of security, and which offered barely any security at all. Had the concept port knocking *relied* on obscurity in hopes to increase security, the attempts to analyse the different mechanisms would surely have been less successful.

The notion of concealment being used for security can be easily compared to Bruce Schneier's quote above about the letter in the safe. If he first gives the safe "along with the design specifications of the safe and a hundred identical safes with their combinations so that you and the worlds best safecrackers can study the locking mechanism", and *then* hides the safe somewhere in New York – he is not trying to obscure the workings of the safe, as he actually submits the safe for peer review, and by hiding his (presumably) secure safe, he is only making it more difficult for an attacker to try and recover the letter within.

Now it should probably be stressed that security through obscurity is not inherently bad! It is only a weakness when it is the *sole* mechanism for security. Assuming a group of the most competent security engineers can design a scheme which is secure, deciding to keep the workings of their scheme a secret does not make their scheme any weaker. Obviously, in reality almost any scheme or implementation has flaws, and the argument of submitting it for analysis by peers is simply that it provides a better chance of finding flaws, should they exist. The primary reason why security through obscurity is undesirable is because it is very difficult, for those who do not know the workings of the security mechanism, to test how strong that mechanism actually is.

The following, taken from Jay Beale's paper on security through obscurity [2, 36], is a good example of true security through obscurity: A company's web server, used to store highly sensitive data, runs the web service on a non-standard port and/or uses long URLs for the content in an attempt to protect the data from being discovered. In this scenario, the sole form of security is through obscurity, which is bad, and bypassing the 'security' mechanism yields access to the server's contents. This can be taken in contrast to port knocking, where bypass of the port knocking mechanism does not provide the attacker with any more access than they would otherwise have had, had port knocking not been used.

This is an important aspect of defence in depth, whereby security is applied at various levels in order to minimise the attack vectors available to a malicious user. Port Knocking mechanisms assist in this goal quite effectively (without obscurity) by allowing us to harvest the 'low-hanging fruit' of vulnerabilities, whilst ensuring that compromise of the mechanism itself does not pose any additional threat to the overall security of the system. Even the most basic form of port knocking, albeit completely insecure in many aspects, would be sufficient in thwarting 90% of unskilled attackers (eg. script kiddies and worms).

4.2 Attacks

Attacks on port knocking schemes can be divided into main two categories: 'interception/impersonation' and 'direct attacks'. The former relates to the act of intercepting packets on their way from the client to the server, which requires the attacker to have some amount of access to the network traffic between the client and the server. The latter refers to attacking the server directly, such as through brute force, which can be done by an unprivileged attacker who does not have access to network traffic. The descriptions of the attacks below are on the 'vanilla' versions of both port knocking and single packet authorzation (see Sections 3.1 and 3.2). The analysis of actual implementations in Chapter 6 will cover whether any of these attacks pose a threat.

4.2.1 Direct Attacks

These attacks form part of the more 'realistic' every-day attacks. As described in Section 3.3, the majority of attackers, which include script kiddies and worms, are like the attacker labelled as 'Trudy' and do not have any access to any of the network traffic between the client and the server. In describing these attacks we can assume that the attacker has knowledge of how the port knocking scheme functions.

Brute Force

Basic port knocking is essentially like a password used to pre-authenticate oneself to the server, before being granted access to the underlying service. The password can be of any length, and each knock in the sequence can be anywhere, hypothetically, between ports 1 and 65535. If the attacker knows that a given implementation of port knocking uses knocks containing a sequence of x ports, he can then construct a program that would knock on every possible combination of x ports, and check after each attempt whether or not the target port is open. Assuming the full range of ports are available, there would be 65535^x possible combinations, which quickly amounts to a large number of combinations. That said, the attacker may not necessarily know which port will be opened by the correct knock, and so may have to resort to port scanning the entire port range after every knock attempt (which may, in itself, corrupt the knock).

In his 'Critique of Port Knocking' [40], Arvind Narayanan states:

> "Suppose you decide on a list of 32 valid ports (the current implementation allows up to 256). How long does the port knock sequence need to be? You might think that since each port is a 16-bit integer, you need 8 knocks, so that you get 8*16 bits or 128 bits of security (virtually unbreakable). But since each port has only 32 possible values (5 bits), what you actually get is only 8*5=40 bits of security (trivially breakable)!"

This description seems to imply a naive understanding of what keylengths actually represent. In cryptographic terms, the size of the chosen key must be selected with regard to which algorithm it will be used with. Arjen Lenstra and Eric Verheul's paper on 'Selecting Cryptographic Key Sizes' [35] describes the key sizes recommended for different cryptographic algorithms. More importantly, the strength of a cryptographic key is based on an attacker's ability to perform his attacks 'offline'. This means that the attacker who has captured an encrypted message, can test every possible key to decrypt that message on his own computer without needing to interact with either of the communicating parties. This is not the same with the basic form of port knocking, which is described in Section 3.1, and which Arvind Narayanan describes above. Due to the fact that the key (knock sequence) *must* be sent to the server in order to know if it is valid, the brute force attack must be performed 'online' by sending every possible knock combination to the server – in which case attempting to brute force 2^{40} possibilities becomes far less feasible. As we will see in Section 6.2, single packet authorization schemes are more vulnerable to offline attacks.

There is one counter-argument against brute force attacks against port knocking systems, and it is that such attacks are very 'loud'. This means that it would be trivial to notice when an attacker was attempting a brute force attack due to the fact that it would involve an enormous number of 'knocks' on the firewall. Imagine you and your friends have a secret knock which must be done before anyone is allowed into the house, and one day you start hearing completely random knocks, and *lots* of them. It's obvious that someone may be trying to guess your knock! In a simple example where the knock consists of only 2 ports, assuming the full port range is available, this amounts to $65535^2 = 4,294,836,225$ possible combinations (far less than 2^{40} as described above), which clearly cannot be brute forced without being noticed, not to mention the fact that such an attack would take a very long time. Assuming that each 'knock' is sent every 0.5 seconds (in order to prevent out-of-order delivery), each full attempt would require 1 second, thus attempting every possible combination of knocks would take approximately 136 years.

In order to counter such attacks it would be simple to implement a system, either in the port knocking daemon itself, or within an Intrusion Detection System (IDS), to automatically block further knocks from a host that has attempted more than x knocks[4]. Furthermore, to protect against the attacker spoofing many IP addresses during its brute force attacks, further connections can be blocked once a given number of IPs have attempted more than x knocks. It must be mentioned, however, that this could be used to carry out a denial of service attack against the client(s), by spoofing the clients' IP addresses and brute forcing the server!

[4]Daniel De Graaf [16] presents a simple IPtables ruleset which automatically blocks, for one hour, IPs attempting to connect to an unused port 4 times in a row. http://daniel.6dns.org/info/iptables/portscan

Attacking the Daemon

Port knocking and SPA both require some kind of 'daemon' whose task it is to observe the knock sequences (and authorization packets) in some way, and then execute a given command should a valid 'knock' be received. Due to the fact that the daemon is a piece of software, then it is possible to assume that there might some way to attack *it* directly with the aim of gaining access to the system that way.

However, it is important to point out that a port knocking daemon is not a traditional networking daemon. At no point in the port knocking authentication process does the client actually *connect* to the daemon (obviously since all ports are closed), unlike SSH for example, where the SSH daemon actually accepts incoming connections from clients. This is an important aspect of port knocking implementations, as the ability to connect directly to a port knocking daemon could make it easier to exploit the running process, for example with a buffer overflow, and gain access to the system via remote code execution.

There are two main types of port knocking daemons used in implementations:

- **Log Readers**: Traditional port knocking schemes have the option to read IP headers of the dropped packets in the firewall logs. The firewall logs are in a standard format, and since the daemon is only interested in which IP knocked, and which port it knocked on, it would be difficult to attempt to corrupt the port knocking daemon by crafting a malicious SYN packet.
- **Packet Sniffers**: Most port knocking and SPA daemons, use packet sniffers (such as libpcap) which allow them to read incoming knocks straight off the wire. This is necessary, in the case of single packet authorization, as the daemon must have access to the payload of the knock packet, and not just the headers. Due to the fact that the daemon is reading the actual packet content and then parsing it to obtain the enclosed knock, it could be possible to create a malicious packet with the aim of exploiting the single packet authorization daemon.

It is theoretically possible to exploit the packet sniffer (eg. libpcap), or the part of the firewall which reads packets. However, libpcap, for example, is amongst the most widely used application programming interfaces (APIs) for network packet captures. Due to this, and its open source nature, its source code is constantly reviewed by a very large number of people, and vulnerabilities, if any, are rapidly eliminated.

Furthermore, due to the closed nature of the firewall used in port knocking, it would be extremely difficult for an attacker to perform OS fingerprinting[5] on the server, without having access to network traffic, in order to determine what operating system and/or platform it was running on. This can be a

[5]OS Fingerprinting, also known as TCP/IP stack fingerprinting, is a technique used to determine the identity of a remote host's operating system by analyzing packets from that host [22].

further advantage, because if a vulnerability in a port knocking daemon were discovered, it would be far more difficult to exploit that vulnerability if the attacker did not know the operating system and/or platform of his target. That said, an attacker could simply try to guess what OS is running, or try a number of different exploits in the hope of striking lucky.

Another advantage of port knocking mechanisms is that they are relatively simple by design, and as such, their security can be analysed far easier than a full-blown package, such as OpenSSH, which performs many actions, and so has many lines of code.

4.2.2 Interception and Impersonation Attacks

These attacks require an attacker with heightened capabilities, and require the ablity to either observe or intercept and then replay the original, or modified packets to the server. This is possible, for example, if the attacker sits somewhere on an area of the network through which all of the packets from the client to the server must pass (eg. an Internet Service Provider), or by sniffing packets on a wireless network. From the players outlined in Section 3.3, the attackers that would be involved in the following attacks would be Eve or Mallory. It should also be mentioned that Eve and Mallory can also perform any of the direct attacks mentioned in Section 4.2.1.

Eavesdropping and Replay

As depicted in Figure 4.4, an eavesdropper on the network may have the ability to observe traffic between the client and the server. In some cases this means that the port knocking sequence could potentially be observed, and then replayed by Eve to the server.

With regard to basic port knocking, and even some SPA schemes, this is a real threat as the server has no way of knowing whether or not the incoming knock is a replay, or a legitimate knock coming from a client. In order to defend against replay attacks, the knock must contain some kind of unique value which will never again be used, thus allowing the server to track which packets it has already received, or which packets are old packets that were never received. In entity authentication protocols, such as those mentioned in ISO/IEC 9798, one crucial element that must be included is the notion of 'freshness' which provides a method for checking that a message was recently generated.

The two most common ways to provide freshness is by using time stamps (clock-based or logical), or nonces (numbers used once). It is also essential to protect the integrity of freshness elements by using some sort of MDC[6]. With timestamps the server can easily see if a packet is old by comparing the time in the knock (or authorization packet) with the time on the local machine. By using a nonce from a large enough space, the server can be relatively confident

[6] A Manipulation Detection Code (MDC) is a block of data that is appended to the end of a message. This can be a MAC or a simple hash can be used and then encrypted together with the message.

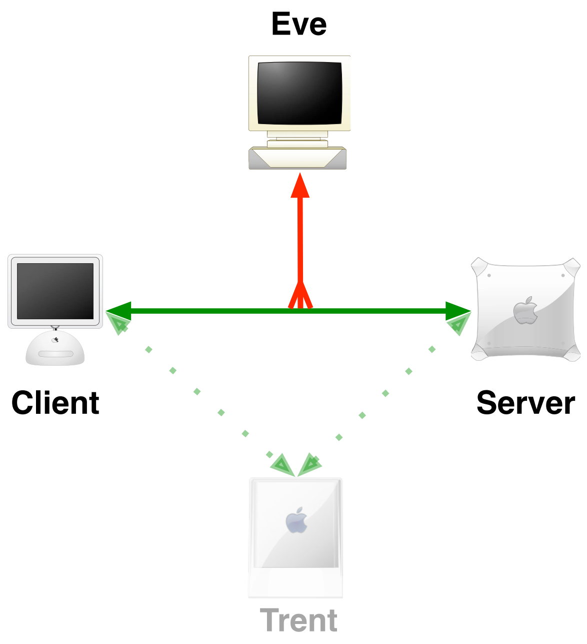

Figure 4.4: Intercepting Knocks through Passive Eavesdropping

that the same number will not occur again anytime soon. Some implementations, such as fwknop (see Section 6.2), use both, although if the timestamp is not checked then it partially defies the point of including it.

A passive attacker may also come across encrypted packets or knock sequences, which can be attacked offline (locally) in the hopes of discovering the password or key used to encrypt (see Section 5.1.5). If the correct key is discovered, then the attacker can craft his own valid authorization request, and gain access to the server.

Man In The Middle

Man in the middle (MITM) attacks are amongst the most difficult to defend against in network security. In this scenario, Mallory has the ability to intercept, modify, block, etc, any network traffic from the client to the server which passes through him (see Figure 4.5). Being in this position offers more sophisticated channels of attack, some of which can be executed in real-time. In its most 'powerful' form a MITM attacker has the ability to spoof traffic to either side of a network conversation, masquerading as the other entity.

Man in the middle attacks on port knocking and SPA are primarily implementation-based, especially considering that both schemes use very different delivery mechanisms. For example a static port knocking system using OTK (One-Time Knocks) is theoretically secure against replay attacks by a passive adversary, but an active adversary such as Mallory can simply intercept the static knock and forward it to the server, thus gaining access for himself. On the other hand, an implementation which binds the authorization request to the client's IP, begins to complicate the task of performing a successful replay attack, as the attacker would need to be able to connect to the server from the client's IP.

There are a number of ways that a MITM attacker can attack a particular implementation, including spoofing public IP address requests, DNS requests, and network time protocol requests to name a few (see Sections 5.1.1, 6.1.1, 6.2.1).

Due to the unauthenticated nature of post-authentication connection attempts (see Section 5.1.2), the simplest attack that an active attacker could perform is to wait for the client to open a port on the server, at which point the attacker can impersonate the client and connect to the server. This attack essentially circumvents the port knocking layer, however, the attacker will still have to authenticate with the underlying service or attempt to exploit it (see Chapter 4 and Figure 4.3). This is not a weakness in the port knocking or SPA schemes, as these kinds of 'session hijack' attacks can be used against numerous protocols.

In the end, a powerful attacker in a MITM position has the ability to fully impersonate both the client and the server, including the ability to send packets with a source IP identical to that of the client or the server, thus making it impossible for either party to know that they are actually communicating via someone else. The only way to possibly defend against such attacks if for all post-authentication connections to be associated with the initial authentication in some way, see Section 5.1.2 for a discussion of some possible solutions.

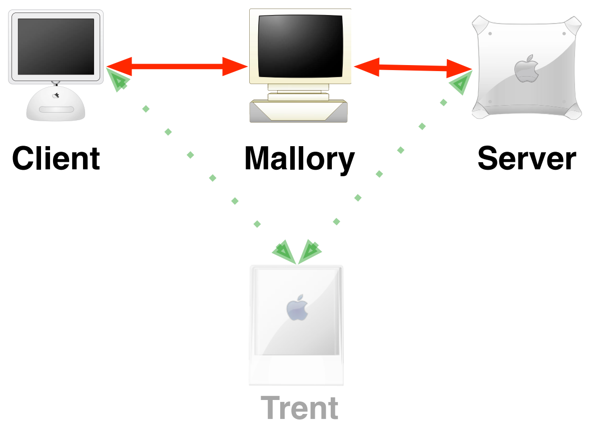

Figure 4.5: Port Knocking Man-in-the-Middle Attack

4.2.3 Methods for Detecting Port Knocking

In order to begin attacking port knocking systems, attackers must first have the ability to detect when a port knocking system is in place. Naturally, this should be difficult in most circumstances as one of the aims of port knocking is that of concealment. An attacker in an non-privileged position (such as Trudy, see Section 3.3), would have a hard time figuring out that a host is protected by a port knocking mechanism, by using purely network-based approaches.

It is important to outline how different port knocking schemes could be detected, not only from an attacker's perspective, but also from a defensive perspective. Standard port knocking would probably be detected by most IDS's as it appears (and functions) very similar to port scans. However, detection can be avoided by increasing the time between sending packets. Most port knocking implementations wouldn't mind if the full knock took 30 seconds as long as all packets arrived in the correct order. Many IDS's would not raise a flag in this case.

On the other hand, single packet authorization mechanisms employ a UDP packet with a small, seemingly random payload. Many protocols use UDP to communicate, and where encrypted traffic is concerned, it is difficult for IDS's to differentiate malicious from acceptable traffic. One method that could be recommended for detecting SPA-like activity would be to watch for single packets (UDP, ICMP or even DNS) which are shortly followed by an actual TCP connection. However, not only would this represent a much heavier load for the IDS, but are also many ways that this could be defeated by using time delays or even instructing the server to connect back to the client (which can be done to bypass certain firewall rules).

Implementation-specific versions of SPA could be tracked using an IDS by programming it to recognise the distinguishing characteristics of the packets created by those implementations. Due to the fact that most SPA payloads are seemingly random, it is difficult to obtain an exact match, however, an example for detecting fwknop packets is given in Section 6.2.

Part III

Real-World Port Knocking

Chapter 5

Practicalities, Limitations and Improvements

5.1 Practicalities, Limitations and Improvements of Port Knocking Mechanisms

There are certain practicality issues when using port knocking in the real world. In all of the examples included, the client and the server have been on the same (local) network. Obviously port knocking mechanisms are more interesting, and indeed more useful, when used to access remote machines. The first significant issue which applies to both port knocking and SPA, is the issue of Network Address Translation (NAT) and its effects on the authorization granted to the client.

5.1.1 Network Address Translation

Network Address Translation is used to share a single public IP address amongst a private network of machines, and as such, each computer on the local network appears with the same IP on the Internet. Due to the fact that the port knocking daemon must open the requested port for a specific IP, this raises two sub-problems. Firstly, the knocking client must provide the server with its *public* IP address. The daemon cannot set a rule in the firewall to allow network traffic coming from the Internet with a private IP like 192.168.1.10. With this in mind it is clear that the client is left with the not-so-simple task of having to discover its public IP address... securely. The main way to discover one's public IP is simply to ask another server on the Internet what IP they see when you connect to them[1]. If an attacker can send a spoofed reply containing his IP address, then the client's knock will enable the attacker to connect the server.

One solution to this issue, when the client resides in a controlled NAT'ed environment, is to set up a simple server on the machine which sits at the boundary of the private network, and this device will usually know its public

[1]One popular website which provides this service is www.whatismyip.com, which can be used by fwknop to discover its public IP in a NAT-situation.

IP. By querying the boundary device, the client can securely discover its public IP. Although the queried IP could still be compromised by an attacker sitting on the same local network, there is less point as both the client and the attacker will have the same public IP (unless the attacker wants to open a port on the server to an external IP).

Another solution, suggested by Rennie deGraaf, John Aycock and Michael Jacobson Jr. [14], can be implemented with SPA schemes. In particular fwknop (see Section 6.2) which could be modified to perform a challenge-response (unilateral or mutual) authentication similar to that described in ISO 9798-4.

They propose a solution to the NAT problem which deviates slightly from the standard 'silent server' setup found in port knocking and SPA. The solution involves a three-pass mutual authentication protocol (whereas port knocking and SPA were provided only unilateral authentication until now), although their protocol assumes that the client will only send certain pre-configured access requests. Below is a slight modification to their protocol which allows for dynamic access requests to be sent:

1: A → B: $req, N_B, MAC_{K_{AB}}(req, N_B)$
2: B → A: PID_A, $MAC_{K_{AB}}(N_B, PID_B, PID_A), N_A$
3: A → B: $MAC_{K_{AB}}(N_A, PID_A, PID_B)$

Where:

A is the client
B is the server
req is the requested access[2].
PID_X is the public IP address of host X.
N_X is a nonce sent to host X.
K_{AB} is a key shared by A and B.
MAC is a cryptographic message authentication code.
, (a comma) represents concatenation.

This mutual authentication protocol helps resolve the issue of public IP discovery on the client side, whilst ensuring that no single 'authorization packet' is generated which could potentially be replayed by an attacker. However, nothing stops a privileged attacker from spoofing the client's IP address and gaining access once the client has performed valid authorization exchange. This problem is due to the lack of association between the authentication session and the subsequent connection (see Section 5.1.2).

However, even once the issue of IP discovery is put aside, we are left with the issue of who is actually authorized at the end of the port knocking process. For example, if you are in a cybercafe and wish to access one of your servers. Once you have completed the knock the server will open a hole in the firewall for your IP, but if NAT is in use at the cybercafe, everyone else will also have access to the server. Of course in actual use, implementations attempt to deal

[2]In [14], the server must be pre-configured with a set of authentication requests, and thus, the requests do not need to be integrity protected.

with this issue by setting a very low timeout before the port is closed, leaving only a small window during which the client (or anyone else) can connect. It may be useful to consider closing the port immediately after a connection is made to the requested port (depending on the protocol used).

5.1.2 Authentication-Connection Association

Up until now we have observed port knocking as performing a simple action, authenticating the client who knocked. This process can be achieved successfully, but what about subsequent connections? It is important to point out that there is no formal association between the client who knocked and the client who is actually attempting to connect to the opened port. So a successfully opened port can then be hijacked by an attacker with the ability to impersonate the client (see Section 4.2.2). A number of possible solutions exist, some discussed briefly by deGraaf et al [14].

Protocol Wrapping and Sequence Numbers

One way to protect a successful authentication from being used by an attacker is to wrap post-authentication connections within an encrypted session. By using a key shared only by the client and the server, the attacker, who does not possess the key, would have no way of crafting valid packets. Invalid packets received on the server-side could be dropped before reaching the underlying service, thus ensuring that only clients with valid keys could get through to the protected service. IPSec or SSL could be used to provide authentication at the TCP/IP layers. Of course this would require some kind of 'wrapper' program which would handle the encryption and verification of post-authentication connections. Not to mention that the added cryptographic operations will undoubtedly increase the load on the server.

Another possibility is the idea of generating TCP Sequence Numbers (SN) in a randomised and authentication-dependent manner, thus allowing the server to verify whether the connection is coming from the client who previously authenticated itself with the port knock or SPA authorization packet.

In their paper [14], deGraaf et al propose that an Initial Sequence Number (ISN) be transmitted in the initial authentication phase, so that the server will know what to look for in a subsequent connection attempt. Although this will work in implementations that do not open ports based on the client's IP (ie. they open the port to any IP), or when the client is behind a NAT, an active attacker in a MITM position can simply intercept the client's connection attempt, steal the ISN, and use it on his own spoofed packet to the server. The benefit of their technique, however, is that it does not require any modification to be made in the way the client and server check sequence numbers of incoming packets.

For the most part these techniques add a layer of complexity on top of the initially simple concepts of port knocking and SPA. Both of these would also require the client to have a heightened privileges in order to manipulate TCP/IP packet fields, which is possible in the case where the client is an admin connecting remotely from his own laptop, but less feasible where the client is sitting at a computer in an internet cafe. The problem lies, in this case, not with the authentication mechanisms provided by port knocking or SPA, but in the fact that TCP connections cannot natively, by design, be bound to a previously authentication session.

5.1.3 Out-of-Order Delivery

The Internet is a vast network of varying latencies. Although basic port knocking may work flawlessly in local network tests, the delays inherent in network communications may be a problem when attempting to use the mechanism in practice. Port knocking requires that the knock sequence arrive in the correct order for the sequence to decode properly. If a single knock arrives out of order, the sequence will be broken and the server will not perform any action, ultimately resulting in a denial of service. Similarly, should an unprivileged attacker, such as Trudy, aim to perform a DoS on a particular client, he can simply send one packet per second to a random port on the server, spoofing the client's IP address as the source address. The server will be unable to differentiate between the attacker's and the client's packets and so the sequence will be broken and authentication will fail.

Another suggestion by deGraaf et al [14] introduces the idea of dividing the bits representing the port number, into *data* bits and *sequence number* bits. In such a way it becomes possible for the server to reorder packets correctly before decoding the knock sequence. This technique would even defend against an attacker trivially spoofing knock packets, in an attempt to cause a DoS, as the server would be able to determine that the attacker's knocks are invalid, and thus discard them. Of course an active attacker like Mallory can easily break such a system, but then again, an active attacker has many other options at hand for performing DoS attacks.

5.1.4 Single Shared Secrets and Multiple Users

Many port knocking and SPA implementations available at the moment only support the use of a single shared secret value, thus meaning that each user is given equivalent rights, or there is only a single user supported. In port knocking schemes a knock can easily take 10 seconds to complete, and it is crucial that all packets arrive in order (see Section 5.1.3). The question of what happens should two users knock at once must not be overlooked. Luckily, there is a simple solution to this problem.

By tracking the knocks received by IP address it is possible to build a 'chain' of the ports knocked on by any number of client IPs. This way, the port knocking daemon is able to differentiate between knocking clients and authorize each one in turn as the correct knock sequence is received and decoded. If a port knocking implementation does not track knocks by IP address, then two clients knocking concurrently will result in a DoS for both of them, as each others' knocks will break their respective sequences. Naturally, SPA schemes do not have this problem as authorization material is sent in a single packet.

In the case of implementations which involve encrypting (or hashing) the knock (or authorization data) using a shared secret, we are presented with an additional overhead on the server-side as it does not know *which* client is authenticating (without the client sending a username along in clear text). The server must then resort to attempting every possible password in the server's user database and hope that a valid knock or authorization packet is decrypted. Once a valid plaintext is decrypted, the server can proceed to carry out the actions specified in the knock (authorization packet) or configuration. It can be questioned, however, whether or not this brute force method would be suitable for use on a system with many users. An attacker may try to use this to bog down the server with decryption attempts by flooding the server with 'junk' ciphertext (spoofing many IPs if necessary).

One solution to this problem, with single packet authorization, is to use a single strong secret (ie. cryptographic key) to encrypt the authorization data, include a username (sent in cleartext) inside the authorization packet, and include that user's password in the hash of the authorization data. The server knows which key to use for decryption, as the username is supplied. This method increases the complexity of the system, but partially reduces the vulnerability of attack from external users to internal users. It has now become unfeasible to brute force the encrypted packet (whereas a password-based encryption key would have been prone to dictionary attacks - see Section 5.1.5).

Another solution is to use public key cryptography, whereby each user has his own private key which can be used to sign the authorization data when the data is encrypted with the server's public key. Public key cryptography, such as GPG, provides data origin authentication and, where applicable, non-repudiation. The use of GPG keys may simplify the issue of multiple users by allowing the server to easily determine which client is requesting authorization simply by decrypting and verifying the ciphertext. Due to the fact that only one encryption key (the server's public key) is required. It is trivial to transmit information securely to the server without having to resort to the brute-force method involved with symmetric keys, as described above.

5.1.5 Password-Based Cryptographic Keys

One simple fact exists in the realm of information security: if humans could remember 32-byte values, authentication would be a much simpler process. The fact of the matter is that passwords are used in a large number of authentication mechanisms, which may make the underlying mechanism prone to dictionary or brute force attacks. The simple example below demonstrates how a basic SPA scheme (described in Section 3.2) can be attacked using a simple dictionary attack.

The authorization data is obtained by hashing the three fields (timestamp, client IP and password) together, using MD5, to produce:

MD5 ("200608062127:192.168.1.100:secretpassword") = 9c6f2af8e1a0f841467f0a1de39f53c8

An attacker with the ability to sniff network traffic from the client to the server, such as Eve, may be able to obtain this 'authorization hash'. One she has the hash she can mount a dictionary attack by simply guessing the user's password and recalculating the hash. The timestamp included in the hash can be guessed by looking at the local clock (assuming both the client and attacker have the correct time set), and the IP address included in the hash can be recovered from the IP header of the client's authorization packet. The attacker now has two out of three of the values needed to calculate the correct hash, so she can simply try to 'guess' the last value, the client's password, until it produces a hash that matches the one contained in the client's authorization packet.

This attack can also be applied to port knocking schemes, although, due to the 'single packet' nature of SPA, it is much easier carry out on these types of schemes. Fwknop uses Rijndael encryption instead of hashing, but the same principle applies. Section 6.2.2, where I present a proof-of-concept tool called fwknop_da, goes into more detail as to how this attack is carried out.

Chapter 6

Implementation Analysis

6.1 Port Knocking Perl Prototype (PKPP)

Martin Krzywinski [32, 33], who coined the term “Port Knocking”, also created the first prototype of such a system in Perl[1]. Obviously, this implementation is a form of traditional port knocking (where the client sends a different packet to each port in the knock sequence), but with a twist. The ‘vanilla’ flavour of port knocking as described in Section 3.1 would rarely be used in real life[2], simply due to the fact that its capabilities are extremely limited, and anyone with access to the knock sequence would immediately be able to replay it to the server and gain access.

A good definition of a port knocking scheme, as created by Martin Krzywinski, is “a method for delivery of information via closed ports” [36]. The notion of “delivery of *information*” is crucial, as it is this that allows a static knock, to be turned into a dynamic knock which actually *provides* the server with specific information relating to the knock and the access being requested. Krzywinski’s Perl prototype achieves this by encoding the port knock data onto the range of admin-selected ports which will be used for knocking. Due to the fact that the port number field in TCP packets is 16-bits long, the maximum amount of data that can be transmitted per knock packet is 2 bytes.

The PKPP allows the admin setting up the system to define what fields a valid knock will consist of. A simple knock will include: the sender’s IP, a port number (to be opened), optional Flag values, optional time-based (or random) values, and a checksum (consisting of the sum of all fields in the knock mod 256). For example:

Client IP: 192.168.1.10
Port: 22
Flag0: 30
Checksum: 167

[1]Perl is an interpreted programming language with features inspired from many other languages - http://www.perl.com

[2]Unless a One-Time Pad of knocks were used, or we weren’t too concerned about the presence of eavesdropping/active attackers.

Produces the following knock sequence: 192 168 1 10 0 22 30 167

The resulting knock sequence is then mapped onto the admin-defined port span. There also exists the option to encrypt the original knock sequence using Blowfish, a 64-bit block cipher with a keylength of 448-bits, and a password-derived key, to protect the confidentiality of the knock. If the plaintext knock sequence (above) is encrypted using Blowfish in CBC mode, then the ciphertext is then converted to bits and mapped onto the port span. Allowing unencrypted knock sequences with this implementation is very dangerous as it would be quite easy for any attacker to have a given port opened for his IP simply by crafting a valid knock in a similar fashion to the one described, there is no secret value included in the unencrypted knock sequence.

The knock daemon has the ability to read knocks out of the firewall log, or directly off of the wire using libpcap. Due to the way that this implementation deals directly with the bit-representation of the knocked ports, it would be quite difficult to compromise the daemon itself with maliciously crafted packets.

6.1.1 Security

The major differentiating factor between this form of 'original' port knocking and single packet authorization schemes, is the nature in which information is transmitted to the server. As explained above, information is actually encoded into strings of bits which can be represented as decimals that are used to knock on specific ports on the server. In some ways this makes the mechanism extremely noisy, yet also quite stealthy. Knocks are more likely to be seen as a random port scan, as they are sent as SYN packets. Although this brings attention to the knocking process, it may also go unnoticed since the packets themselves have no payload. A knowledgeable attacker eavesdropping on the network traffic would notice that port knocking is in use as soon as a network connection is opened to a seemingly non-existent host.

From a protocol perspective, there are points pertinent to the security of this implementation. Firstly is the very basic checksum which is obviously only there to allow the server to check whether a properly formatted knock was decoded. Such a checksum does not provide true integrity protection and cannot be used to track which knocks have already been used, in order to protect against replay attacks. Martin Krzywinski recommends that the knock be encrypted, and this would indeed provide a higher level of protection against analysis. Knowing the port span being used by a given server is essential to using the system, but also when attempting to decode intercepted knocks. Given some time and a given amount of observed knock sequences it should be quite easy to determine the port span, although it seems that the best port span to use would be many small non-contiguous ranges, as this would make it harder for an attacker to determine exactly where each range starts and ends.

If the attacker knows the port span then he could easily run a dictionary/brute force attack against the ciphertext (see Section 5.1.5), simply by checking the resulting plaintext for the presence of known values. For example, if the client knocks on the server and then connects to SSH port 22 from its IP, the attacker knows that the client's IP and the value 22 must be present somewhere within the plaintext. Although in this implementation the format of knocks can be defined by the administrator, the IP of the knocker (eg. 192.168.1.10) will be quite unique in a knock string. That said, it may be slightly more difficult to automate this in a live attack, as there are no 'static' values as-such in a port knock. Port knocking uses no particular field delimiters, nor any static strings which could be identified in the plaintext. An incorrectly decrypted plaintext could still be parsed normally by reading the binary bits and converting them back to decimal values. If the attacker knows that the 'checksum' is used in the knock and its supposed position in the knock sequence, he could check whether a given plaintext is valid. Alternatively the attacker could simply assume that the client is requesting authorization for its IP, and search the plaintext for the client's IP.

The primary concern with this implementation is that of replay attacks. Although there are some anti-replay features, some of them require that state be maintained on both the server and client side. This is especially inconvenient for the client side where the knocking may not always be performed from the same machine. If only time-based features are used, however, and if two servers are running the same configuration of port knocking, an attacker could intercept the knock for one server and replay it on the other. No indication of destination is included in the knock which would alert the server if it was being sent a knock destined for another server.

If this implementation is used with full timestamps (for replay protection), One Time Knocks using incrementing flag values (also for replay protection), a strong cryptographic key (ie. 448 random bits - with Blowfish), and a large port span, then this implementation becomes more resistant to replay attacks but starts to become more bloated in terms of knock length.

Drawbacks

This implementation can only really be used with encryption enabled, as unencrypted knocks offer no protection against replay attacks, and contain no 'secret' value meaning that anyone could potentially send a valid knock and gain access. However, when encryption is used, the knock sequences can become quite long (eg.16 knocks), especially if a small port span is used. This makes the port knocking process extremely loud and obvious, thus making it much easier to detect through simple traffic analysis. Combined with the issue of out-of-order delivery which is inherent in port knocking, it is essential to ensure that there is enough time between knocks to compensate for network delay. In total, a knock can take approximately 10 seconds to execute which may or may not be acceptable in certain circumstances, or to certain users.

With regard to the network 'noise' involved with port knocking, it may be an issue that port knocking may experience problems when used in conjunction with some form of IDS. Knocks can easily be mistaken for port scans, thus resulting in the client's IP being temporarily blacklisted and a denial of service will occur.

Due to the fact that information is transmitted through the port numbers (and only 2 bytes of data can be transmitted per packet), it is essential to keep the knock sequence as short as possible to ensure that knocks do not take too long. Unfortunately due to the message expansion involved in the use of asymmetric cryptography, it is not practical to use GPG[3] to encrypt the knock sequences with the PKPP, as the resulting knock sequence would be extremely long. There would be several benefits to using public keys to produce the knock sequence. Firstly the knock sequence would provide origin authentication of the knocker, and supporting multiple users would be a simpler task.

6.2 fwknop - FireWall KNock OPerator

Firewall Knock Operator (fwknop) is the SPA implementation by Michael Rash [45, 46, 47, 49]. Fwknop (also programmed in Perl) used to be a port knocking implementation, but with the appearance of the single packet authorization concept, its primary mode of operation is now SPA over UDP (although TCP and ICMP are also supported). From the couple of SPA implementations that were tested, fwknop definitely seems to be the most flexible in terms of how the system can be set up. Fwknop, as with most SPA implementations, sets up a knock string which is then packed into a UDP packet, utilising the payload available, and sent off to the server. In SPA the 'knock' is called an authorization packet.

In fwknop, the authorization packet consists of five pieces of data:

```
Random data:   3765661773077220 (16 bytes of random data)
Username:      admin (local username)
Timestamp:     1155665551 (local timestamp)
Version:       0.9.7 (fwknop version)
Action:        1 (access or command mode)
Access:        10.0.0.1,tcp/22 (desired access or command)
MD5 sum:       qZayT4TNoK1pCOI66Wr0oA
```

The MD5 is calculated over all of the fields in order to allow the server to check whether the authorization data was received correctly. The Username, Access and MD5 sum fields are Base64 encoded. These strings are then concatenated together, delimited by colons (:), to form the plaintext authorization data:

```
3765661773077220:YWRtaW4=:1155665551:0.9.7:1:MTAuMC4wLjEsdGNwLzIy:
qZayT4TNoK1pCOI66Wr0oA
```

The resulting string is encrypted using Rijndael, in Cipher Block Chaining (CBC) mode, with a block size of 128-bits and a key length of 256-bits. By encrypting the authorization string, that contains a hash of the data, the plaintext becomes integrity-protected, and thus an attacker would be unable to

[3]GNU Privacy Guard is an open source implementation of Phil Zimmermann's popular PGP (Pretty Good Privacy) hybrid public key crypto-system. See Section 2.2.2.

modify the plaintext (through ciphertext manipulation) without those modifications being detected. The cryptographic key in fwknop's default configuration, however, is derived from a user-entered passphrase (8-16 characters in length[4]) through a series of MD5 hashes[5], meaning that there are only approximately 95^{16} (100-bits) possible keys (there are 95 printable ASCII characters, and a maximum of 16 characters per password), far less than the full 256-bits of a Rijndael key. The encrypted authorization data is then packed into a UDP packet and sent off to the specified server.

On the server, the fwknop daemon (fwknopd) watches on UDP port 62201 (by using libpcap to sniff the wire) for inbound authorization packets. When a packet is received fwknopd attempts to decrypt it, parse the decrypted data by finding the colon (:) delimiters, and checks that the MD5 hash of the fields matches the MD5 hash included in the packet. The daemon also has the ability to use p0f[6] to perform passive OS fingerprinting of the client, and can be configured to only allow access to certain types of machines. Although this can easily be circumvented, the attacker must first know which types of machines are being granted access, which would again require access to network traffic to fingerprint a valid client.

As stated in Section 4.2.3, detecting SPA packets is more difficult than detecting port knocks as SPA packets are less obvious than long and 'loud' port knocks, and contain seemingly random data which can potentially blend in with normal network traffic. In the case of fwknop, however, there are some indicators which could be programmed into an IDS to flag that fwknop *may* be in use on the network. Fwknop packets go to port 62201 by default, so a rule could be defined to watch for traffic to that port, although this would easily be circumvented by changing to a custom port. Alternatively, an IDS can attempt to track fwknop packets by checking the data length of the UDP packets. An fwknop packet encrypted using Rijndael will be anywhere between 80 and 160 bytes in length. A packet encrypted using GPG will be anywhere between 500 and 1600 bytes in length [50]. Although a quick test of such a rule on my machine, running several common applications, reveals that about 15 packets meet the above requirements every 10 seconds. Obviously this rule would generate far too many false positives. Thus, the only way to successfully track for fwknop packets would be to track packets of the lengths mentioned above, going to UDP port 62201 of a machine.

6.2.1 Security

From an entity authentication perspective, fwknop is basically an implementation of ISO/IEC 9798-2 mechanism 1, a one-pass method for providing unilateral authentication using timestamps and encryption, which has been applied

[4]If the user-entered passphrase is less than 16 characters long then it is padded with zeros until it is exactly 16 characters long.

[5]The Perl Crypt::CBC module is responsible for turning the user-entered passphrase into a 256-bit key suitable for use with the Rijndael algorithm, by running the passphrase through a series of MD5 hash operations.

[6]p0f is a popular tool developed by Michal Zalewski which performs passive OS fingerprinting. http://lcamtuf.coredump.cx/p0f.shtml

to the concept of 'firewall authentication' (although fwknop does not follow the actual standard). This provides effective authentication of the client requesting authorization to the server, and helps protect against replay, although the timestamp used in fwknop is not actually checked against the clock on the server. Both the timestamp and the random data fields are used as fresh values or 'randomisers' which help ensure that each authorization packet will yield a unique MD5 sum which can be logged to check for replays.

However, in this lack of timestamp checking exists a possibility for a man in the middle attack where the attacker intercepts and blocks an authorization packet from the client. Due to the fact that the packet never reaches the server, its MD5 hash will not be logged, and the timestamp will not be checked for freshness at the time of delivery. Although a replay of this intercepted packet at a later date will only open the requested port for the IP included in the packet (probably the client's IP at the time), this can still be used to the attacker's advantage should he have the ability to connect from the same location as the client, or masquerade as the client on the network.

These issues of 'block and replay' attacks can be reduced by implementing a 'window of acceptance' against which the timestamp is checked before any action is performed. The window of acceptance can be set to compensate for standard clock drift, although a client with incorrect time may experience a denial of service as the client's timestamp will not match the valid time on the server when the timestamp is checked. By checking the timestamp and making sure that it's within the window of acceptance, the server is able to reject any potential replays that are outside the accepted timeframe.

Fwknop does not include the identity of the server in the authorization packet, which means that two servers with an identical setup (same users and keys), are vulnerable to replay attacks of each others' authorization packets - especially if timestamps are not checked. An attacker could simply intercept a valid authorization packet on route to one of the servers, and immediately (or at a later date, if no timestamp check is implemented) replay it to the second server.

Network Address Translation

If the client is on a network that uses NAT, then it must use some mechanism in order to discover its public IP address (see Section 5.1.1). Fwknop has the option to do this (`--whatismyip`) by accessing www.whatismyip.com[7] and parsing the returned IP address. The attacker can use this to his advantage by spoofing any requests that the client sends out to discover its public IP. The result of this attack is that the client will use the spoofed IP address in the authorization packet, hence, by intercepting and blocking this packet, the attacker has successfully obtained a free pass to open the requested port for an IP of the attacker's choice. Similarly, fwknop allows the client to enter a

[7]Although fwknop uses whatismyip.com as the default, it can be configured to use any other website which returns the IP address in the title of the web page.

hostname[8] to which access must be granted. Beware when using this feature as the fwknop client first performs a DNS lookup to obtain the IP address of that hostname, which it then puts into the authorization packet. Spoofing DNS requests quite a simple attack, and an attacker could simply return the IP address of his choice which would then be the one used in the authorization packet, resulting in the same compromise as mentioned above.

Fwknop also has the option (`--source-IP`) to instruct the server to grant access to the source IP of the authorization packet (in cases where the client cannot successfully discover its public IP address). If an attacker is able to intercept and block a packet constructed in this way, then he has obtained a free pass to grant access to any IP he wants, at any time, without performing any spoofing of any kind! For a live attack, the attacker can simply modify the source IP of the authorization packet as it passes through his network, and the server will grant access to an IP of the attacker's choice, and not the client's. This option should be avoided as it is the least secure (especially without timestamp checks), and manually setting one's external IP in fwknop should be done wherever possible[9].

For the sake of completeness it should be mentioned that a MITM attack could still be performed to obtain a valid authorization packet for a specific date/time if the client uses the Network Time Protocol (NTP) to update its local clock. By spoofing the client's NTP requests, together with spoofing the client's public IP requests (where applicable), the attacker is able to intercept and block a valid authorization packet which would later authorize access to the server at the date/time and from the IP of his choice!

Also note the fact that using fwknop to run commands on the server, as opposed to opening ports on the firewall, is mostly immune to the issues of NAT and to some extent replay attacks as well. An observer would have no way of knowing what action was carried out on the server[10], thus making client-spoofing and 'block and replay' attacks useless.

Weak Passwords

As mentioned previously, due to the fact that weak passwords can (and often are) be used to generate the cryptographic key used to encrypt fwknop packets with Rijndael, the default configuration is quite vulnerable to a simple dictionary attack. Section 6.2.2 describes this attack and presents a tool (fwknop_da) that I designed for this purpose, which can crack any fwknop authorization packet encrypted, with Rijndael, using a dictionary word in less than a minute. Although using a 16 character passphrase containing the full

[8]A hostname is a unique name used to identify devices on a network. Each hostname is assigned to an IP address. For example www.apple.com is the hostname for the IP 17.112.152.32. Some services, such as DynDNS.org provide

[9]The fwknop help recommends that the `--whatismyip` option be used instead. Although this method can still be spoofed by an attacker, as detailed above, it is significantly better than using `--source-IP`.

[10]When an SPA packet is sent to open a specific port, the attacker immediately knows that access was being requested for that port from the client's IP.

uppercase, lowercase, digits and symbols would probably be sufficient to defeat a dictionary attack, it would still be significantly less than the full 2^{256} possibilities of a Rijndael key. I have already recommended that fwknop be enabled to use random 256-bit Rijndael keys, as this would ensure that any kind of brute-force attack be unfeasible, although it would make key management a more complex issue, as most people cannot remember 256-bit values.

It should be mentioned, however, that fwknop does have one major advantage over the PPKP. Thanks to the larger payloads available in UDP packets[11], fwknop supports asymmetric encryption through GPG which creates significantly larger ciphertexts. This allows strong encryption to be provided together with entity authentication and non-repudiation. Due to the public-key nature of GPG, each user possesses their own public/private key pair, so it is easy to verify who has sent a particular GPG signed and encrypted message, and also gain the assurance that only that person could have sent that message (as only they possess their private/signing keys). This effectively eliminates the issues revolving around multiple-users, and also eliminates the issue of dictionary attacks on weak passwords.

Anonymity

Fwknop has recently gained the ability to send authorization packets over the Tor onion network [18, 49]. Although using this feature requires the daemon to run an actual listening TCP socket for the client to connect to[12], this mechanism loses the design goal of concealment, but gaining the anonymity offered by the Tor network. The Tor network functions by sending encrypted packets over a random path through several servers (onion routers). The key element is that no individual router knows the complete path through the onion network (the network of onion routers) from the client to the server (see Figure 6.1). This makes it impossible to track exactly where a packet is coming from, and thus traffic analysis becomes a much more difficult task. It would also be possible to make use of Tor's 'hidden service' feature to conceal which server a client is sending authorization packets to. As it is no longer possible to track the origin of packets, and all traffic must routed over the Tor network, an attacker who used to be in a MITM position can no longer carry out any of his attacks against the client nor the server.

Due to the way Tor pre-establishes TCP connections for the client, this makes it unsuitable for port knocking schemes which require the ability to send SYN packets to a non-listening host. It is no longer possible to perform 'data transmission across closed ports', which is the way that port knocking schemes, such as PKPP, function.

[11]Remember that port knocking mechanisms can only transmit 2 bytes of data per 'knock' packet. See Section 6.1.

[12]Tor does not allow UDP or ICMP packets to be sent through the Tor network. Only once the Tor exit node establishes a TCP connection with the server (see Section 2.1.3), does Tor start routing data from the client over the established TCP connection to the server [18].

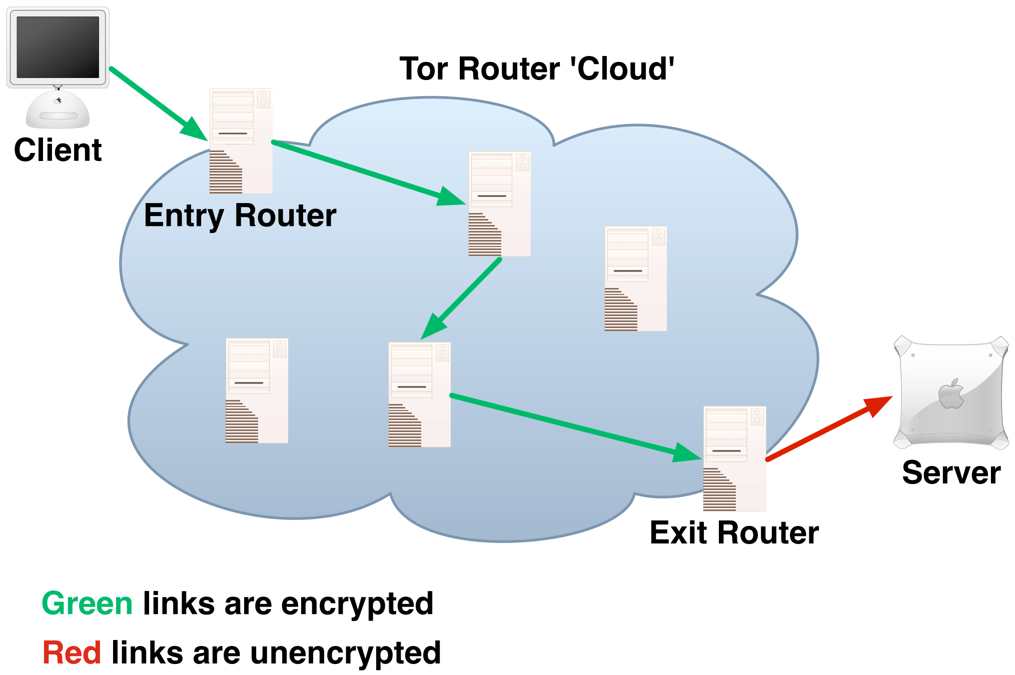

Figure 6.1: Routing Data over the Tor Onion Network

Drawbacks

The main drawback to fwknop seems to be the way that cryptographic keys are generated (when using passphrases), and multiple users are handled using symmetric encryption. Although fwknop makes a good attempt at managing multiple users, the implementation still requires that received authorization packets be brute forced. Apart from being a less-than-elegant way of managing multiple users, it is unsure, without further testing and benchmarking, whether or not this issue would have a large impact on a server used by many users (for example to protect the VPN ports of a large company's Virtual Private Network[13]).

In order to help reduce the risk of dictionary attacks, without resorting to the use of GPG, it may be beneficial to allow users to use randomly generated Rijndael keys, as previously mentioned. Finally, it is important to fix the lack of timestamp checks in fwknop, to greatly reduce the possibility of 'block and replay' attacks.

[13]A Virtual Private Network is a private communications network used to protect the confidentiality of data flowing from a remote user to the internal network of an organisation.

6.2.2 Dictionary Attack on fwknop

As described in Section 6.2, fwknop is primarily vulnerable to dictionary or brute force attacks on the default configuration.

A passive or active attacker can intercept the authorization packet $f(P, D)$ where P is the password used to encrypt (or be hashed with) the authorization data D. A basic offline dictionary attack can be performed by the attacker by testing a list of possible passwords P', and comparing the value $f(P', D)$ with $f(P, D)$. If the two values match, then the correct password has been discovered. This is known as the password-authenticated key exchange problem, which has been pondered over for over a decade [5, 4, 8].

To illustrate such an attack in progress I developed a proof of concept tool called `fwknop_da` (dictionary attack), which takes in a fwknop ciphertext from the command line, the network, or from a file full of ciphertexts, and uses a user-defined dictionary file to attempt to recover the plaintext authorization packet. If this is successful, the attacker can essentially authenticate himself to the server and instruct it to open any port or perform any command (depending on the configuration of the server and what ports/commands are enabled for the client).

As described above, this script tries each password in the password file in succession, using Rijndael, and employing the same passphrase-padding used in fwknop (see Section 6.2). The script tests each decrypted plaintext by searching for an fwknop version number. If this is found then the correct password has been found, and the plaintext authorization data can be parsed. This tool could be expanded to recalculate the MD5 of the fields and compare it to the MD5 contained in the packet.

The original dictionary file used to run this attack was 40Mb in size and contained just under 4 million words (there are just under 1 million words in the English language). However, due to the fact that fwknop only accepts passphrases between 8 and 16 characters long, any shorter or longer words were removed from the dictionary leaving us with a 30Mb file containing 2.8 million words.

Figure 6.2 shows a sample run of fwknop_da running in network mode, intercepting an encrypted authorization packet, and testing many passwords before finding and parsing the correct plaintext. This simple tool tests approximately 2600 words per second on an Apple Powerbook G4 1.67 GHz, and over 5,000 words per second on an Apple MacBook Core Duo 2.0 GHz (utilising only one core). Assuming a constant rate of 5,000 words per second, it would take approximately 560 seconds, or just under 10 minutes, to try every possible word in the 30Mb dictionary file! Of course the dictionary file can be improved by adding commonly-used numbers and symbols in the place of certain letters, whilst still keeping the time required to try every combination relatively low. It should also be mentioned that brute-forcing all 95^{16} (100-bits) possible passphrases in such a way is not feasible, without a more dedicated and optimised implementation.

After some further testing, I discovered that fwknop can be modified to

```
$ perl fwknop_da.pl -b -d 1

[+] Starting fwknop_da.
[+] ** Running in debug mode 1 **
[+] ** Running in benchmark mode **
Path to Wordlist: /all2

[+] Listening for knocks on network (port 62201)...

[+] Received message: U2FsdGVkX19cfskOKeZ01VpD2Fo5qNcGeZ2R1h/+o
FQEzwzMPJd0/JYv58Ewixml7pc/BHBAHsGaxaOeRYCiDFhBfaPYRJJijeDGotl0
BnZ2pQjsYnqoTsZ8CMZDJocy7jcEz9SMC12cBky9JKm9vg

[+] Starting Dictionary Attack...
10000 words tried.
20000 words tried.
30000 words tried.
40000 words tried.
50000 words tried.
60000 words tried.
70000 words tried.
80000 words tried.
90000 words tried.

[+] Password found: indoxylsulphuric
[+] 92436 words tried.
[+] Knock Version: 0.9.7
[+] Decoded message: 8121860749824944:admin:1156789640:0.9.7:1:
                     192.168.1.1,tcp/22:Gk7819P3dRn273h/XDVV7Q
[+] Packet fields:
        Random data: 8121860749824944
        Username:    admin
        Remote time: 1156789640
        Remote ver:  0.9.7
        Action type: 1
        Action:      192.168.1.1,tcp/22
        MD5 sum:     Gk7819P3dRn273h/XDVV7Q
[+] Time elapsed: 35 seconds (2641 words/sec).
```

Figure 6.2: Sample run of fwknop_da

fully support arbitrary-length passphrases, which would allow users to use long passphrases, significantly reducing the threat of dictionary attacks[14]. Although this greatly reduces the threat of dictionary attacks, a memorable passphrase would have to be 39 characters long before it was of similar strength to a 256-bit key[15].

Another way to slightly improve passphrase security could be to combine the user's username together with the passphrase that is used to derive the encryption key. Since the username is not used as a public identifier in this case, it may as well be used to reinforce passphrases. This is essentially like 'salting' the password and means that any given password produces a vast amount of different cryptographic keys. In theory this would square the difficulty of running a dictionary attack on keys generated in such a way. In practice, however, one would find that common usernames can probably narrowed down to a short list, especially if we know some information about the users of the system (such as their names)[16].

[14]Since passphrases are a combination of multiple words (potentially dictionary words), a exhaustive search for every possible 'phrase' would require a larger amount of time and effort.

[15]A randomly-generated 256-bit key would still be significantly harder to brute force than a memorable 39-character passphrase.

[16]Common usernames would be attempted first, for example: admin, administrator, firstname.lastname (if we know the users of a system).

Chapter 7

Further Research

7.1 Port Knocking in Malware (Backdoors)

The concealment aspect of port knocking is a feature which may be of interest to malware writers, especially when a newly-opened port may be indicative that a host has been compromised and is running a 'listener' to allow the attacker to connect to the machine. Similarly, an open port may allow other attackers to discover the already-compromised machine and claim it as their own. Worms spread automatically, sometimes by scanning the local network (or the Internet) and connecting to vulnerable services running on discovered machines in order to exploit them (exactly one threat that port knocking aims to protect against), and it is in the worm owner's interest to maintain control over his network of compromised computers.

By implementing a form of port knocking into their worms, the authors can maintain control over machines that become compromised. This is already believed to be in use in trojans and worms, although no actual malware has actually been found to contain a port knocking implementation yet. There do exist two implementations of port knocking that are particularly well suited for use in malware. Both of these implementations allow for the server to listen for a valid knock before opening a backdoor server for an attacker to connect. SAdoor and its predecessor cd00r [12, 9], both have this functionality with, and are particularly well suited for use as backdoors as opposed to normal port knocking mechanisms.

It would be interesting to see whether port knocking becomes more widespread in malware, as the malware authors continue to try and conceal the presence of their programs. The simplicity of most port knocking implementations would easily allow for such mechanisms to be used in malware which is limited in size.

7.2 Port Knocking in Enterprise Environments

Port knocking and SPA remain relatively novel ideas which have not been fully considered by the security community, and most implementations currently available are proof-of-concepts, with very few exceptions. Due to this, users in enterprise environments may be reluctant to adopt these mechanisms to help protect some of their vital servers for fear of unwanted side-effects or excessive server loads. Port knocking and SPA have probably never been tested to actively protect a system with more than 10 users, and if they have, the results of such trials would be highly interesting.

However, limiting the use of such authentication mechanisms to the purpose of administrating servers, for example to allow admins to open port 22, or to allow them to run commands directly on the server without having to connect to them, is entirely within the scope of the current implementations.

As Dawn Isabel mentions in her analysis of traditional port knocking schemes [28], an important aspect which must be addressed before we can see any widespread adoption of 'firewall authentication' mechanisms, is a certain 'stability' in the implementations which will ensure that the port knocking layer would fail gracefully, ensuring that the outcome of such a failure be acceptable by its users. Port knocking implementations would also have to be checked and re-checked for potential vulnerabilities in the daemon itself, although the programming language used by the implementation would play an important role in the risk of having such vulnerabilities.

Port knocking systems would also have to have a certain degree of audit-ability in order to check whether the system was performing its functions correctly and that nothing out-of-the-ordinary was occurring on the system. As such, it would be important for the mechanism to keep suitable logs for this purpose, something which most implementations do not offer at all.

The important concept of password rotation which is mandatory in many large corporations would also need to be worked into the port knocking schemes in order to ensure a similar assurance of security and to prevent passwords being discovered after a long period of time. In her paper Dawn Isabel also presents the idea of using the company's pre-existing LDAP directory to provide an interface for changing passwords to port knocking accounts. This would, however, add another attack vector where "an attacker [...] could simply compromise a less secure application to obtain a valid single sign-on password" [28].

The use of a single port knocking or SPA host at the boundary of a company's network may be a unique way to allow remote users to access several different services by enabling the port knocking server to open and forwarding ports from the outside to specific hosts on the inside of the network. Obviously there are many issues which would need to be resolved before such mechanisms could be widely used, however, none these issues are not due to any obvious weaknesses the port knocking or SPA mechanisms, and it is a matter of time because a unique solution is presented which offers to address many of the points raised in this thesis.

Chapter 8

Conclusion

During my analysis of port knocking as a network security mechanism it has been become clearer how it must be viewed. Many of the criticisms about port knocking come from those who are looking for the be-all end-all of network authentication mechanisms, no wonder they're disappointed. When looking at port knocking schemes it becomes clear that certain issues crop up time and time again.

What is port knocking? Port knocking is essentially an extension of defence in depth - another layer in our security onion - and another tool in the security specialist's box of tricks which aims to perform a very specific task. That aim, from a bird's-eye view is to harvest a large number of 'low hanging fruit' in terms of host-based vulnerabilities whilst offering extra services in the process.

Although the design aim of concealment is not a necessity in port knocking, it is a significant advantage when it comes to protecting a host running services. By hiding services in such a way, they can be protected from attacks against which there may not exist a simple defence. Port knocking schemes also help to reinforce the notion of least privilege by removing access from those who have no reason having access in the first place, attackers in all forms. By authenticating a client before granting access, there is a significant improvement in security over no-authentication access that exists without port knocking.

The two primary concerns that remain with port knocking schemes are denial of service and man in the middle attacks. As with any protocol, there are indeed ways to perform DoS attacks on port knocking, although in certain cases this may actually be preferable as it results in a more secure machine. Should port knocking fail, however, the failure itself will not result in compromise of the underlying host and the attacker will have to attack the underlying service just as he would have if port knocking had not been used. Replay attacks are possible to protect against using both port knocking and SPA implementations, and it is important to remember that even if a replay attack is successfully carried out, the firewall will only be opened for the original client's IP, and not the attacker's IP, thus eliminating all but the highly skilled and privileged attackers.

It is clear that implementations currently have an issue with the way cryptographic keys are generated from passphrases, however, with the use of randomly generated full-length keys, or asymmetric cryptography such as GPG, it is possible to use strong, encrypted, authentication protocols. On critical systems, or even small systems which receive very little use, it can be argued that port knocking provides useful protection against a variety of everyday attacks, whilst giving users the freedom to use the server as they normally would. Ultimately, implementing port knocking is a trade-off of effort against the benefits gained. Is would be essential to take into account the computational overhead required in running such a system on a large scale.

Even the most basic 'single packet port knock' mechanism, albeit insecure in many ways, would be adequate to protect against a large majority of everyday attacks, such as script kiddies and worms. For example a simple iptables rule, such as this one [16] by Daniel De Graaf, requires that a secret port be knocked on before port 22 is opened to the client IP[1]. Admittedly part of the bonus of using a 'firewall authentication' scheme is partially due to the relatively low number of hosts that implement such schemes at this point in time. It is important to remember that staying one-step ahead in the security domain can make an important difference to the security of one's systems.

By taking advantage of tried and tested entity authentication protocols, SPA schemes such as fwknop can be reasonably confident that the design goal of authentication will be carried out successfully. The increased amount of data which can be transported in an SPA packet allows for cryptographically-strong authentication data to be sent to the server, without worrying about out-of-order delivery, or lengthy knock-times. Similarly, the use of public key cryptography may help to simplify such mechanisms in multi-user environments, especially where a PKI system is already in place.

One significant issue that seems to exist in both port knocking and SPA schemes, however, is the issue of Network Address Translation. Not so much when it comes to the client discovering it's public IP, but more so the notion of who actually has access to the server at the end of a successful authentication. The lack of association between the authentication and subsequent connections is as-of-yet an unresolved issue which, in the presence of a privileged attacker, may result in unauthorised access being granted to him.

Although 'firewall authentication' schemes such as port knocking and SPA may have some outstanding issues, it is my opinion that these schemes do an excellent job of reducing the number of threats, and thus the risks, to a service-running host. A large majority of attackers would be thwarted using these mechanisms, and should the popularity of port knocking and SPA increase significantly in the future, I strongly believe that bypassing these schemes will prove difficult regardless. Most importantly port knocking and SPA provide a realistic defence against 0day attacks, which is not something that can be found in very many, if any, security mechanisms currently available.

Port knocking and SPA seem particularly well suited to administrative ac-

[1]Trivially sniffable and replayable, although his implementation does protect, somewhat, against port scans and brute force attacks.

cess, or protecting services which are particularly vulnerable to various kinds of attacks - all while allowing a maximum amount of flexibility to the system's users. All-in-all I feel that port knocking and SPA are worthwhile additions to defence in depth, which require a minimum amount of effort and resources to set-up and maintain. Hopefully these schemes will become continue to grow in popularity, so that their security can be more widely tested (which is a requirement for any new security mechanism), and hopefully improvements can be made that will justify their place as the outermost layer of the security onion.

Out of the implementations I have seen, fwknop seems to provide the most robust mechanisms and the most comprehensive set of features. Although it still supports its previous mode of traditional port knocking, which may be appealing to use in some scenarios, SPA is now the default mechanism. With support for both symmetric and asymmetric encryption, users can choose to use the mode that is best suited to their needs. I have recommended several fixes and improvements to Michael Rash's implementation, most of which are mentioned in this thesis, which would reinforce the security of fwknop . The fact that fwknop runs in Perl allows the client utility to run on a large number of platforms, although the fwknop daemon can only run on Linux-based hosts due to the requirement for IPtables.

Bibliography

[1] Barham P. et al (2002) ‘Techniques for Lightweight Concealment and Authentication in IP Networks’. Intel Research Berkeley. July 2002.
Available at: `http://www.intel-research.net/Publications/Berkeley/012720031106_111.pdf`

[2] Beale, J. (2000) ‘“Security through Obscurity” Ain’t What They Think It Is’ [Online].
Available at: `http://www.bastille-linux.org/jay/obscurity-revisited.html`

[3] Bejtlich R. (2006) ‘Single Packet Authorization with Fwknop’. TaoSecurity, August 21, 2006.
Available at: `http://taosecurity.blogspot.com/2006/08/single-packet-authorization-with.html`

[4] Bellare M, Pointcheval D, Rogaway P. (2000) ‘Authenticated Key Exchange Secure Against Dictionary Attacks’. Lecture Notes in Computer Science.
Available at: `http://www.iacr.org/archive/eurocrypt2000/1807/18070140-new.pdf`

[5] Bellovin S, Merritt M. (1992) ‘Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks’. Proc. of the Symposium on Security and Privacy, pages 7284. IEEE, 1992.
Available at: `http://www.ussrback.com/docs/papers/cryptography/neke.ps`

[6] Bishop M. (2005) *Introduction to Computer Security.* Addison Wesley, Pearson Education.

[7] Borss C. (2001) ‘DROP/DENY vs. REJECT’. Listserv post to Braunschweiger Linux User Group (lug-bs@lk.etc.tu-bs.de).
Available at: `http://www.lk.etc.tu-bs.de/lists/archiv/lug-bs/2001/msg05734.html`

[8] Boyko V, MacKenzie P, Patel S. (2000) ‘Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman’. Lecture Notes in Computer Science.
Available at: `http://www.iacr.org/archive/eurocrypt2000/1807/18070157-new.pdf`

[9] FX. (2000) ‘cd00r: not listening remote UN*X shell’. [Online].
Available at: `http://www.phenoelit.de/stuff/cd00rdescr.html`

[10] Computer Emergency Response Team Coordination Center (CERT-CC) (2002). ‘Overview of Attack Trends’. Carnegie Mellon University.
Available at: `http://www.cert.org/archive/pdf/attack_trends.pdf`

[11] Computer Emergency Response Team Research (CERT) (2005). ‘CERT Research 2005 Annual Report’. Carnegie Mellon University.
Available at: `http://www.cert.org/archive/pdf/cert_rsch_annual_rpt_2005.pdf`

[12] CMN. (2003) ‘SAdoor: A non-listening remote shell and execution server’. [Online].
Available at: `http://cmn.listprojects.darklab.org/`

[13] Cormen T, Leiserson C, Rivest R, and Stein C. (2003) *Introduction to Algorithms*. McGraw Hill, MIT, 2003.

[14] deGraaf R, Aycock C, and Jacobson M. (2005) ‘Improved Port Knocking with Strong Authentication’. *ACSAC 2005*, pp. 409-418.
Available at: `http://www.acsa-admin.org/2005/papers/156.pdf`

[15] deGraaf R, Aycock C, and Jacobson M. (2005) ‘Improved Port Knocking with Strong Authentication’. [Presentation]. Department of Computer Science, University of Calgary.
Available at: `http://pages.cpsc.ucalgary.ca/~degraaf/papers/portknocking-presentation.pdf`

[16] De Graaf, D. (2006) ‘complex’. [Online].
Available at: `http://daniel.6dns.org/info/iptables/complex`

[17] DiGioia P. (2004) ‘Behind Closed Doors: An Evaluation of Port Knocking Authentication’. Donald Bren School of Information and Computer Sciences, University of California, Irvine.
Available at: `http://www.ics.uci.edu/~pdigioia/publications/essays/Port%20Knocking.pd`

[18] Dingledine R, Mathewson N, Syverson P. (2004) ‘Tor: The Second-Generation Onion Router’. [Online]. The Freehaven Project. May 2004.
Available at: `http://tor.freehaven.net/cvs/doc/design-paper/tor-design.html`

[19] Doyle M. (2004) ‘Implementing a Port Knocking System in C’. Department of Physics, University of Arkansas.
Available at: `http://portknocking.sourceforge.net/files/Implementing%20a%20Port%20Knocking%20System%20in%20C.pdf`

[20] ElGamal T. (1985) ‘A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithm’. IEEE Transactions on Information Theory, v. IT-31, n. 4, 1985, pp469472 or CRYPTO 84, pp1018, Springer-Verlag.
Available at: `http://crypto.csail.mit.edu/classes/6.857/papers/elgamal.pdf`

[21] Ferreira A. (2006) ‘Coarse Port Knocking’. [Online]. Accessed: January 2006.
Available at: `http://coarseknocking.sourceforge.net/`

[22] Fyodor (1998) ‘Remote OS detection via TCP/IP Stack FingerPrinting’ [Online]. Insecure.org
Available at: `http://insecure.org/nmap/nmap-fingerprinting-article.txt`

[23] Google Groups (2006) 'block_ssh_guessers'. April 16 2006.
Available at: `http://groups.google.com/group/comp.os.linux.security/browse_thread/thread/30c8a88ddeed53dc/b75ee069451189fe?#b75ee069451189fe`

[24] Graham-Cumming J. (2004) 'Practical Secure Port Knocking'. *Dr. Dobb's Journal*, November 2004, Issue 366, pp. 51-53.
Available at: `http://www.ddj.com/184405890`

[25] Hou JC. (2004) 'Port Knocking'. [Presentation]. Department of Computer Science, University of Illinois at Urbana Champaign.
Available at: `http://lion.cs.uiuc.edu/courses/cs397hou/lectures/PortKnocking.ppt`

[26] Internet Assigned Numbers Authority (2006) 'Port Numbers' [Online].
Available from: `http://www.iana.org/assignments/port-numbers`

[27] Internet Assigned Numbers Authority (2006) 'ICMP Type Numbers' [Online].
Available from: `http://www.iana.org/assignments/icmp-parameters`

[28] Isabel D. (2005) 'Port Knocking: Beyond the Basics'. GIAC Security Essentials Certification (GSEC). *SANS Institute*, March 9, 2005.
Available at: `http://www.sans.org/reading_room/whitepapers/sysadmin/1634.php`

[29] Kung L, Hou JC. (2004) 'CS397 Network Systems Lab Project 5: Port Knocking'. Department of Computer Science, University of Illinois at Urbana Champaign.
Available at: `http://lion.cs.uiuc.edu/courses/cs397hou/project5.pdf`

[30] Krivis S. (2004) 'Port Knocking: Helpful or Harmful? An Exploration of Modern Network Threats'. *SANS Institute*, May 2004.
Available at: `http://www.giac.org/practical/GSEC/Stuart_Krivis_GSEC.pdf`

[31] Klima V. (2006) 'Tunnels in Hash Functions: MD5 Collisions Within a Minute'. Cryptology ePrint Archive, Report 2006/105.
Available at: `http://eprint.iacr.org/2006/105`

[32] Krzywinski M. (2003) 'Port Knocking: Network Authentication Across Closed Ports'. SysAdmin Magazine, pp 12:12-17.

[33] Krzywinski M. (2003) 'Port Knocking'. *Linux Journal*, June 2003.
Available at: `http://www.linuxjournal.com/article/6811`

[34] Krzywinski M. (2004) 'A Critique of Port Knocking - Author's Response' [Online].
Available at: `http://www.portknocking.org/view/about/critique/`

[35] Lenstra A.K, Verheul E.R (2001) 'Selecting Cryptographic Key Sizes'. Crypt, vol 14, no 4, 2001.
Available at: `http://www.win.tue.nl/~klenstra/key.pdf#search=%22Selecting%20Cryptographic%20Key%20Sizes%22`

[36] Maddock B. (2004) 'Port Knocking: An Overview of Concepts, Issues and Implementations'. *SANS Institute*, September 2004.
Available at: http://www.giac.org/practical/GSEC/Ben_Maddock_GSEC.pdf

[37] MadHat Unspecific and Simple Nomad (2005) 'SPA: Single Packet Authorization'. [Presentation]. BlackHat Briefings 2005.
Available at: http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-madhat.pdf

[38] Martin K. (2004) 'Click on this, you muthas'. *The Register*, February 2004.
Available at: http://www.theregister.co.uk/2004/02/23/click_on_this_you_muthas/

[39] Menezes A. J., Oorschot P. C. V., und Vanstone S. A. (1997) *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL. 1997.

[40] Narayanan A. (2004) 'A Critique of Port Knocking'. *Newsforge*, August 2004.
Available at: http://software.newsforge.com/software/04/08/02/1954253.shtml

[41] National Institute of Standards and Technology (1995) FIPS PUB 180-1. 'Secure Hash Standard'. National Institute of Standards and Technology, 100 Bureau Dr. Stop 8900, Gaithersburg, MD 20899-8900.
Available at: http://www.itl.nist.gov/fipspubs/fip180-1.htm

[42] National Institute of Standards and Technology (2002) FIPS PUB 180-2. 'Secure Hash Standard'. National Institute of Standards and Technology, 100 Bureau Dr. Stop 8900, Gaithersburg, MD 20899-8900.
Available at: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

[43] National Institute of Standards and Technology (1999) FIPS PUB 46-3. 'Data Encryption Standard (DES)'. National Institute of Standards and Technology, 100 Bureau Dr. Stop 8900, Gaithersburg, MD 20899-8900.
Available at: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[44] National Institute of Standards and Technology (2001) FIPS PUB 197. 'Advanced Encryption Standard (AES)'. National Institute of Standards and Technology, 100 Bureau Dr. Stop 8900, Gaithersburg, MD 20899-8900.
Available at: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[45] Rash M. (2004) 'Combining Port Knocking With OS Fingerprinting'. ;login: The USENIX Magazine. December 2004, Volume 29, Number 6, pp 19-25.
Available at: http://www.usenix.org/publications/login/2004-12/pdfs/fwknop.pdf

[46] Rash M. (2006) 'Advances in Single Packet Authorization'. [Presentation]. SchmooCon, January 2006.
Available at: http://www.cipherdyne.com/fwknop/docs/talks/shmoocon2006_fwknop_slides.pdf

[47] Rash M. (2006) 'Single Packet Authorization with Fwknop'. ;login: The USENIX Magazine. February 2006, Volume 31, Number 1, pp 63-69.
Available online (subscription): http://www.usenix.org/publications/login/2006-02/pdfs/rash.pdf

[48] Rash M. (2006) 'Maximum Netfilter'. [Presentation]. OSCON, July 2006.
Available at: `http://www.cipherdyne.com/fwknop/docs/talks/oscon2006_slides.pdf`

[49] Rash M. (2006) 'Service Cloaking and Anonymous Access; Combining Tor with Single Packet Authorization (SPA)'. [Presentation]. DEF CON, August 2006.
Available at: `http://www.cipherdyne.com/fwknop/docs/talks/dc14_fwknop_slides.pdf`

[50] Rash M. (mbr.at.cipherdyne.org). 30 August 2006. 'Re: Interview'. [Personal Correspondence].

[51] Rivest R (1990) 'Cryptography'. From the Handbook of Theoretical Computer Science, edited by J. van Leeuwen, Elsevier Science Publishers B.V., 1990.

[52] Rivest R, Shamir A, Adleman L. (1978) 'A Method for Obtaining Digital Signatures and Public-Key Cryptosystems'. Communications of the ACM, v. 21, n. 2, Feb 1978, pp. 120-126.
Available at: `http://theory.lcs.mit.edu/~rivest/rsapaper.pdf`

[53] Rivest R. (1993) 'The MD5 Message-Digest Algorithm'. Networking Working Group, Request for Comments: 1321.
Available at: `http://theory.lcs.mit.edu/~rivest/Rivest-MD5.txt`

[54] Schneier B. (1996) *Applied Cryptography: Protocols, Algorithms, and Source Code in C (Second Edition)*. John Wiley & Sons.

[55] Shimonski R.J. et al. (2003) *Best Damn Firewall Book Period*. Syngress Publishing.

[56] Slashdot (2004) '"Port Knocking" for Added Security'. Accessed: 01/2006
Available at: `http://slashdot.org/it/04/02/05/1834228.shtml?tid=126&tid=172`

[57] Slashdot (2004) 'Port Knocking in Action'. Accessed: 01/2006
Available at: `http://it.slashdot.org/article.pl?sid=04/04/14/1832222`

[58] Slashdot (2004) Author: Michael Rash. 'Combining Port Knocking With OS Fingerprinting'. Accessed: 01/2006
Available at: `http://it.slashdot.org/it/04/08/01/0436204.shtml`

[59] Slashdot (2005) Author: Michael Rash. 'Going Beyond Port Knocking; Single Packet Access'. Accessed: 01/2006
Available at: `http://it.slashdot.org/article.pl?sid=05/05/30/1128209&tid=172&tid=106`

[60] Stallings W. (2003) *Network Security Essentials: Applications and Standards*. Prentice Hall.

[61] Tan CK. and Capella. (2004) 'Remote Server Management using Dynamic Port Knocking and Forwarding'. Special Interest Group in Security and Information Integrity (SIG$\hat{2}$).
Available at: `http://www.security.org.sg/code/sig2portknock.pdf`

[62] TBONIUS. ‘Introduction to Port Knocking’. *Section 6*.
Available at: `http://www.section6.net/wiki/index.php/Introduction_to_Port_Knocking`

[63] Wang W. et al (2005) ‘Finding Collisions in the Full SHA-1’. Shandong University, Jinan 250100, China.
Available at: `http://www.infosec.sdu.edu.cn/paper/sha1-crypto-auth-new-2-yao.pdf`