Journal of Information Assurance and Security.
ISSN 1554-1010 Volume 9 (2014) pp. 001-008

Port Knocking with Single Packet Authentication using Asymmetric Key Cryptography

Michael Reeves

Mike.reev0@yahoo.com

***Abstract*: Protecting services from attack is the sole purpose of a firewall; however, some services (generally those for remote administration) require enough leniencies in their rules that the protection of a firewall is quite limited. Port Knocking is a method which may helps protect against attack, by preventing firewall responses to connection requests until appropriate conditions are met, in the form of a knocking packet sequence. To harden the security of Port Knocking further, asymmetric cryptography can be used to reduce the number of knocking packets to a single packet while also authenticating the individual user.**

***Keywords*:** Port Knocking, Cryptography, Authentication, Firewall, Security

I. Introduction

As reliance on computers and the Internet have increased over the years, so too have cyber attacks. In order to combat these attacks and protect against exploitation of vulnerabilities within the technology so heavily relied upon, computer security has become an essential component to networking and system administration. A layered security approach, consisting of security controls at various points through a network, has been found most suitable to protect vital resources and services, but in-turn adds complexity to the existing infrastructure.

The network boundary is a critical point in the security of a network's infrastructure, as it is the first layer of defense. It is standard practice to use a firewall to limit both inbound and outbound traffic. By blocking traffic to services, the risk of attack is mitigated since the service is inaccessible to those who might exploit it. However, there are some cases in which external access is necessary.

When allowing inbound traffic through a firewall, it is prudent to restrict the traffic to only known-authorized sources. Insufficiently restricted firewall filters result in potentially giving an attacker the foothold they need to launch attacks. Yet, there are some cases when the source information needed for the firewall filter cannot be known in advance. For this reason Port Knocking was created—to provide a mechanism for dynamically generated one-to-one firewall rules.

II. Port Knocking

A. Port Knocking Overview

Port Knocking is a security mechanism first made famous by Martin Krzywinski [1][2]. The objective of Port Knocking is to allow firewall ports to be dynamically opened or closed through the use of specifically crafted packets. A Port Knocking scheme is only possible through the use of two agents: the knocker and the listener. The listener resides on the system with the protective firewall, traditionally using the firewall logs as a means to listen for potential knocking traffic. The knocker resides on the remote client and is the mechanism to send the specifically crafted packs. Both the knocker and the listener must already agree and understand the knocking pattern.

By definition, the knocking traffic is a covert channel—a means of communication through a media outside its intended use [3][4]. Covert channels can be seemingly anything in which both communicating parties have access. For instance, two parties wishing only to answer questions with either a "yes" or a "no" may utilize the file-lock feature of a shared file. If the file is in file-lock state the answer is "no," whereas, if the file is open for writing the answer is "yes." As such, the question may be asked in the open (accessible to prying eyes and ears), but only those who know how to get the answer understand the more critical portion of the communication.

Port Knocking uses a similar communication system. A Port Knocking configuration begins with the firewall set to deny all inbound traffic [1][2]. However, this configuration also means all services are cut off from the Internet—including to authorized users. To finish the configuration, a sequence of ports is decided upon, in which if the listener observes the port sequence it will then open a pre-determined port. The knocker is then configured for the same sequence.

An obvious use for Port Knocking is for securing remote administration. Secure Shell (SSH), for instance, is a common service used in remote administration, which is normally bound to port 22. One could bind SSH to another port, but the fact of the matter is that the protocol could easily be identified simply by its response to a connection request [3]. Another option for protecting a server running SSH is to create a firewall rule which limits connections only from the IP addresses known to be used by remote administrators.

However, due to the dynamic nature of IP address issuance by Internet Service Providers (ISP), it is not always possible to know which IP address the administrator is using, so it may not be possible to create a sufficiently restrictive firewall rule. With Port Knocking, the SSH port can be blocked at the firewall until the knocking client has sent the correct port sequence. Likewise, another port sequence may be used to covertly inform the listener that SSH port is no longer needed and may be blocked by the firewall yet again.

B. Port Knocking Example

Pork Knocking can be configured to nearly any imaginable configuration. To exemplify one possible configuration, we will use the aforementioned securing of SSH. The sequence we will employ is ports 43, 65,010, and 198 (as depicted in Figure 1).

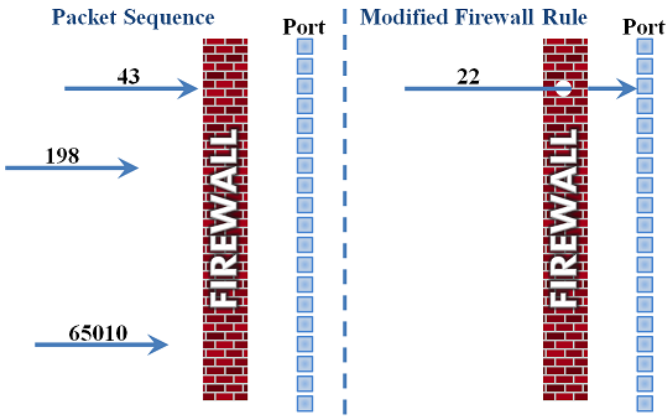

Figure 1. A Port sequence triggering listener to open port 22

The firewall is configured to block all inbound traffic, including ports 43, 198, and 65,010. Once the correct sequence is blocked by the firewall, the listener acknowledges that the knock sequence occurred and dynamically creates a firewall rule which allows port 22 from the IP address that generated the knocking sequence. Traditionally, a separate knocking sequence is used to trigger the listener to remove the previously created firewall rule; however, more advanced listeners can be configured to remove the rule after a specified time [5].

C. Port Knocking Downfalls

Like all security practices, Port Knocking is by no means perfect. One well documented problem is the natural unreliability of packet sequences. Each packet from a source to a destination can take different routes, each potentially traveling at different speeds. Though the speed differences for each packet may be nanoseconds, the difference may be enough to cause one packet to arrive before its former. If this were to occur, though the knocker sent out the correct sequence, the listener sees a different sequence (one that does not match the configuration) and no change to the firewall rules is made, as depicted in Figure 2.

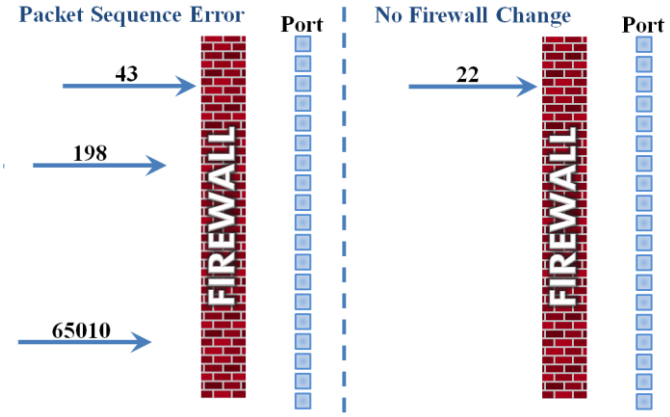

Figure 2. An out of sequence port knock

One simple way to mitigate the sequence issue is to use a sequence consisting of only one knocking packet. The obvious downfall of this approach is that it is less secure. Port Knocking was developed to prevent against port scans from uncovering open ports. The logic behind the practice is that it is highly unlikely that a port scan, even one trying ports at random, would stumble upon the correct sequence (and no additional ports attempted during the sequence) [1][2].

Without the correct sequence, the port scan result would find that no services are running, since all connection attempts are blocked by the firewall. However, with only one port used as the knocking sequence, the listener would create the new firewall rule once the scan tries the knocking port; thus, allowing the scanner access to the servicing port being protected. In such a case, a port scan has a 50% chance of scanning the knocking port before the servicing port; thereby, exposing the protected service.

To reduce the likelihood of a port scan uncovering the correct sequence, the number of ports in the sequence must be increased. The obvious downfall to increasing the number of ports is that it also increases the likelihood of one or more packets arriving out of sequence.

The biggest issue plaguing Port Knocking is the fact that it is simply an implementation of "security through obscurity" by means of covert channels. Just like the covert channel example previously mentioned (in which anyone with access to the file-lock status and knowledge of its significance could in turn understand the message) anyone who can observe the knocking sequence can also replicate it for themselves. A potential solution to this problem is with the employment of cryptography [6][7][8][9]. With encrypted knocking packets, sensitive information (such as port number and action) can be protected against eavesdroppers. Encryption can even be used to protect against replay attacks by including a timestamp and the originator's IP address.

Unfortunately, all previously proposed options utilize symmetric cryptography which has security issues of its own. Symmetric cryptography uses the same key for encryption and decryption; thus, to read a symmetrically encrypted message one must decrypt it with the same key which was used for encrypting [10][11]. The process is mathematically represented as follows:

$$m = D_K\left(E_K\left(m\right)\right)$$

Using Port Knocking with symmetric encryption, at a minimum two systems must have the key: the knocker and the listener [9][10][11]. They both inherently can encrypt and decrypt the message since both transactions use the same key. Because the listener is configured to use the key for decryption, each knocker must be issued the same key. For this reason, all implementations of Port Knocking with symmetric cryptography is not a form of authentication, but rather it is authorization (the belief that if the payload is encrypted with the appropriate key, the knocker must be in the circle of trust and, therefore authorized access to the service port).

Several weaknesses exist with symmetric encryption; most notably key issuance [10]. Because every system requires the same key, there must be a means to distribute the key. Distribution can be achieved in a multitude of ways: sneaker-net, on a controlled network, over the internet through a VPN, etc. All methods have no means of limiting the number of keys created and no means to ensure that once a key is delivered to a knocking agent it is not further distributed. This lack of control is the reason Pork Knocking with symmetric cryptography is authorization-based—the assumption the key has not been provided to unauthorized parties.

Other issues with symmetric cryptography are far more technical than key distribution; pertaining more to the strengths and weaknesses of the protocols in play. The size of the encryption key, for instance, plays a significant role in the security of an encryption algorithm [10][11]. An algorithm's key size (or "key space") is the number of bits used for the key, which in-turn determines the total number of keys that are possible. A 64-bit key has 18,446,744,073,709,551,616 possible values; however, with systems able to try millions per second, finding the correct key is only a matter of time.

Symmetric cryptography uses small keys in order minimize the computational time of the encryption/decryption process [10][11]. There is, however, a threshold where key space is simply too small because computers can be used to attempt every possible key until the correct one is uncovered (referred to as a brute-force attack). The Data Encryption Standard (DES) is one example of a symmetric encryption algorithm whose key size can be uncovered through a brute-force attack within a reasonable period time [12].

Key size is not the only factor when determining if a symmetric encryption algorithm is secure. Not all encryption algorithms are equal. The mathematical principles behind the algorithms must be sound for it to withstand cryptanalysis—the process of uncovering the key by means other than brute-force attacks [10]. Microsoft's LanMan hashing algorithm and NSA's Skipjack algorithm both have fallen victim to cryptanalysis [10][12][13]. For these reasons the Advanced Encryption Standard (AES) has been suggested as an appropriate encryption algorithm, being both mathematically sound and supporting sufficient key sizes [11].

Even with using AES, potential problems still remain. Any symmetric encryption algorithm is susceptible to cryptanalysis if the message being encrypted is smaller in size than the encryption key since the message is a variable of the encryption process [10]. A message shorter than the key size will result in an encrypted output (called the "ciphertext") that lacks randomness, allowing some of the key's bits to be inferred. To mitigate this risk a predetermined random variable (known as a "salt") may be used to add randomness to the message, but the same salt value must be configured on the listener and all the knockers.

One final concern of symmetric cryptography is susceptibility to known-plaintext attack [10]. This attack requires knowledge of the message being encrypted and the ability to see the ciphertext. Because one employing Port Knocking would be concerned with network eavesdroppers, this attack warrants interest. While a "salt" does mitigate a known-plaintext attack to an extent, if a cryptanalyst is aware of what the message should contain, he can in-turn use that information to analyze how the message was altered by the key. Just as having a message with insufficient length lacks in randomness, so too does a message with predictable input.

By observing the knocking packet(s) and subsequently opened port (identified by the next established connection), an attacker can gather the information he needs for a known-plaintext attack [10][14]. Furthermore, adhering to a known standard solidifies the placement of the data within the message, allowing the attacker to focus his cryptanalysis on the predictable portion. While known-plaintext attacks and insufficient message lengths do not reveal the entirety of a symmetric key, they reduce the time needed for a brute-force attack by a factor of two for each inferred bit of the key.

III. Single Packet Authentication with Asymmetric Cryptography

A. Asymmetric Cryptography

1) Overview

Unlike symmetric cryptography, which uses only one key for both the encryption and decryption processes, asymmetric cryptography uses two distinct (yet related) keys [10][11]. The two keys serve different roles. One of the keys, referred to as the "private key," is solely possessed by the owner of the key. If the private key is compromised, all security to be gained from the encryption process is lost [10]. The other key, however, is by design available to the public (and thus known as the "public key").

Where the security of symmetric keys lies in the randomness created by their algorithms, asymmetric cryptography gets its strength through complex computations [10]. Though each algorithm is different, the theory behind any asymmetric algorithm is that the complex computations are too difficult to calculate within a reasonable period of time without both key pairs. By knowing the key pairs, their relationship can be determined, greatly simplifying the calculation.

Where symmetric algorithms can typically be proven and/or disproven to be mathematically sound, asymmetric algorithms rely on the belief that the complex computations cannot be solved using a simpler formula [10]. While it is difficult to prove that there is a way to simplify the complex calculations (which would deem the algorithm insecure), it is impossible to prove that there isn't. In fact, it is always possible that a mathematically finding may cause an asymmetric algorithm to suddenly hold no merit [15].

The sound mathematics in symmetric key cryptography is what allows for such smaller key size [10]. Asymmetric cryptography does not reap the same benefit [10][11]. Because asymmetric cryptography relies on the difficulty of complex mathematics regarding the relationship of the two keys, it in-turn requires very large keys—generally at least 10 times the key size of its symmetric counterpart. One reason for the greater key size is the fact that not every key pair has a mathematical relationship. For instance, an algorithm which uses prime numbers in its computation cannot have a key that is divisible by anything other than itself and one.

Using only 6 bits, a symmetric algorithm would allow for 64 keys to choose from, but of those 64 possible keys only 19 are prime numbers. Of those 19 keys, there are sure to be pairs which simply do not have a mathematical relationship in the asymmetric algorithm. A smaller key size means less possible matching key pairs, and therefore, brute-force attacking every possible matching private key becomes less cumbersome than the complex computation the algorithm uses for its security [10].

The previously stated weakness of symmetric cryptography relating the message size to the key size does not exist in asymmetric cryptography [10]. This weakness exists in symmetric cryptography because of its procedural nature—ciphering/deciphering occurs in a particular sequence using the key as a variable. Because asymmetric cryptography is not based on a procedure but rather a relationship between two keys, the message size cannot be used to reveal any bits of the private key. The trade-off, however, is that asymmetric cryptography is significantly more taxing on the system than symmetric. Because encrypting/decrypting with an asymmetric algorithm is more taxing, it is not recommended for large messages.

The benefits of asymmetric cryptography lie in the fact that only one person has the private key [10][11]. Because anyone potentially has access to the public key, then anyone can encrypt a message using it. An encrypted message can only be decrypted by the owner of the corresponding private key. Using K for the private key and P for the public key, an asymmetric algorithm can be mathematically expressed as follows:

$$m = D_K\left(E_P\left(m\right)\right)$$

On the surface, not much seems different between asymmetric and symmetric cryptography. However, where symmetric cryptography only protects for integrity, confidentiality is additionally enforced through asymmetric cryptography [10][11]. Some may argue that symmetric cryptography has a degree of confidentiality; although, the level of confidentiality would be limited to those individuals who have the encryption key (which at a minimum are two: the knocker and the listener). Conversely, asymmetric cryptography ensures confidentiality down to the individual.

2) Other Capabilities

The real power with asymmetric cryptography occurs when the sequence is reversed. An individual encrypting their message with their private key enables anyone with the public key (potentially anyone in the world) to read the message. Attempting to tamper with the encrypted message or encrypting with a key other than the private key results in the message being corrupted during decryption [10]. If the message is successfully decrypted it must have been encrypted with the corresponding private key, and therefore originating from its owner. This inverse use of an asymmetric algorithm is expressed as follows:

$$m = D_P\left(E_K\left(m\right)\right)$$

Because only one individual has the private key, they personally must have been the one who encrypted the message [10][11].

Encrypting a message with one's private key is a digital signature; however, tradition digital signatures use a similar but slightly more complex application of this approach [10][11]. As mentioned before, asymmetric cryptography is more taxing on a system than symmetric cryptography; therefore, most files are too large to be encrypted in whole. In order to reduce the burden of the encryption/decryption process, traditional digital signatures use the hash value of the file as the message, rather than the original file itself. Through this implementation, the digital signature validates the identity of who signed it; whereas, the hash value validates the integrity of the file.

A successful decryption only proves that the keys are a legitimate pair. The proof that the owner of the private key is whom they claim to be is another matter. The acceptance of a key pair's owner is largely based on trust. Anyone can create their own asymmetric key pair, claiming to be someone they are not. Whether or not one believes them can be determined by who created the key.

Trustworthy asymmetric keys are generated by a Certificate Authority (CA) [10][11]. In addition to creating the keys, the CA also creates a digital certificate; consisting of the public key and the CA's information. The certificate is digitally signed by the CA to prove its authenticity.

The security scheme is quite simple: if one trusts the CA, then they can implicitly trust the public key. If there is no reason to trust the CA, then there is no reason to trust the claimed identity of the certificate (or key pair) they create [10]. Several CAs exist for the sole purpose of issuing certificates which are considered trustworthy. These trusted CAs require applicants to go through a painstaking process to validate the applicant's identity before issuing their certificate. Additionally, the CAs are responsible for issuing certificate revocations in the event a private key is compromised. Lastly, CAs may keep a copy of the private key for key escrow.

Rather than pay for the creation of certificates, networks which want to exclusively use asymmetric key pairs in-house can stand up their own CA. By doing so, each certificate is trusted by its network peers since the CA is locally housed and controlled. Trying to us the certificate outside the network is possible; however, it would likely be ineffective as those outside the network have no reason to trust the CA.

B. Single Packet Authorization

Port Knocking with Single Packet Authorization was first proposed by Michael Rash [8]. The methodology is simple: a single packet contains the user's information (e.g. username and password), local timestamp, action requested (open/close and port number), and a salt. This information as a whole is used to create a hash value. All of this data, including the hash

value, is encrypted with the symmetric key and used as the packet payload.

This system works off the assumption that the symmetric key has not been compromised, either during key distribution or by cryptanalysis. Of course if the key was compromised, then all of the content could be easily changed, making the hash value (contained in the payload) moot to that regard. If the key is not compromised then the hash value and the information used to create it is rendered unreadable to an eavesdropper.

Working with the assumption that they key is not compromised, the listener takes the provided plaintext information and generates a hash value using the same hashing algorithm [8]. If the hash values match, then the payload is likely to not have been altered, and the firewall change is made. If the hash value does not match, then no change is made.

As mentioned before, this method only provides authorization since there is no authentication process in place [8][9]. While a user's information can be passed along this way, anyone with access to the symmetric key may view this information; thus, the confidentiality of the user's information is limited to the circle of trust associated to the symmetric key. To exemplify this, suppose Alice produces her information via single packet authorization while Bob eavesdrops on the traffic. Because Bob is in the circle of trust, and therefore also has the symmetric key, he is able to decrypt and read Alice's information. Bob can then send his own knocking packet with the newly acquired information about Alice. Because the same key is used, as far as the listener is aware Alice is the one requesting access. Both Alice and Bob are authorized to knock and make these changes, but nothing validated that Bob is who he claims to be.

C. Port Knocking using Single Packet Authentication

In February of 2013, I initially proposed the possibility of hardening Port Knocking with asymmetric-cryptography based authentication, which became the basis of this work [16]. Asymmetric cryptography ties a private key to a single owner; thus, authentication is a capability that symmetric cryptography does not have. By utilizing the private key to encrypt data one can ensure the data originated from the owner of the private key. Using the aforementioned example, if Bob encrypted Alice's information with his private key, the listener would know that Bob was the one making the request even though the payload contains Alice's information. Two methods can determine if this has occurred: 1) decrypting the data with Alice's public key or 2) decrypting the data using Bob's public key. If the listener attempts to decrypt with Alice's public key, the message will be corrupted and unreadable. If the listener decrypts with Bob's public key, the message will be intact and prove that Bob is the actual sender.

In order for a single packet to be used for authentication, all pertinent data must be able to be encapsulated in the packet's maximum transmission unit (MTU). The MTU is the maximum amount of data which can be used as packet payload, as defined by the protocol. Exceeding the MTU will result in the data being broken up into multiple packets or outright truncated. In addition, because all packet content is readable (due to the availability of the public key), measures must be made to protect against attack.

A replay attack is one where an eavesdropper records the packet and launches it again at a later time [9]. One measure against a replay attack is to use a timestamp. By synchronizing with the same Network Time Protocol (NTP) server, the knocker and the listener will have the same system clock time (within fractions of a second) [8]. By providing a plaintext timestamp and an encrypted version of the timestamp, one can ensure that the timestamp was not altered; however, it does not prove the packet has been replayed. Since NTP keeps the knocker's and the listener's clocks synchronized, a margin of error can be used to minimize the effects of a replay attack.

The difference between the timestamp on the packet and the current time is the packet's latency. This latency can be used to detour replay attacks. A maximum latency tolerance of two seconds, for example, would provide ample headway for bandwidth issues while minimizing susceptibility of a replay attack. There is still the possibility of a replay attack occurring in the two second window.

By adding the IP address in conjunction to the timestamp, replay attacks are virtually impossible. If the IP address in the packet header does not match the encrypted IP address in the payload, then the packet must be a replay attack. Because no two systems can have the same IP address at the same time, the only way to defeat this approach is for an attacker to position himself between the knocker and the listener (known as a man-in-the-middle), at which time other attacks can be launched.

Network Address Translation (NAT), however, can cause two systems to appear (to the receiver) to have the same IP address [9]. NAT is used to allow systems in a Local Area Network to use one publicly routable IP address. NAT is used in most networks today. As such, a replay attacker and a knocker on the same NAT-enabled network would appear to be the same system from the listener's perspective.

NAT can cause several issues with single-packet authentication. In order for the IP address in the header and encrypted version in the payload to match, the knocker must be aware that they are being NAT'd [9]. The public IP address must be used when creating the encrypted payload, which means the knocker must be able to be configured to encrypt a specified IP address. Michael Rash's single packet authorization aims at addressing the NAT problem by not including the IP address; however, as a result Rash's method is prone to replay attacks [8].

Another security concern is firewall rule piggy-backing. By observing a Port Knocking sequence of any kind, one can infer what port is being opened. Once observed, on subsequent Port Knocking transactions an eavesdropper on the same NAT'd network can send their own packet to the opened port and gain access to the newly opened firewall rule. To protect against rule piggy-backing, the latency of the packet can be used to automatically close the opened port. For instance, a rule can be made to open a port automatically (closing it after twice the length of time as the packet latency). Of course, currently open sessions must be preserved beyond the packet latency.

Where single packet authorization has a means of relaying the port number and action (open/close), it is not recommended to do so with this method. With single packet

authorization, the data is encrypted and only those with the key may decrypt the message to see the port number. With single packet authentication using asymmetric keys, all of the packet's content may be read by any potential eavesdropper. Therefore, revealing the port provides eavesdroppers information that could assist them in a time-of-request/time-of-use attack.

A time-of-request/time-of-use attack takes advantage of the gap between the time a resource is requested vice when it is actually used. In our case, a time-of-request/time-of-use attack gap is the time between when the knocking packet is received (in turn, changing the firewall rules) and when the firewall rule is actually used. The listener acknowledges the knock by allowing the next connection to the protected service within the threshold time. The rule is dynamically created and only allows traffic from the IP address referenced in the knocking packet.

As mentioned before, all traffic from systems on a network utilizing NAT will seem to be coming from the same publicly routable IP address. Working with the assumption that the port and action are contained in the knocking packet, and the knocker is on such a network, any other system which can see the knocking packet can in-turn craft its own connection attempt, effectively beating the knocking system to the punch.

NAT is not the only concern with time-of-request/time-of-use attacks. Any system in which all traffic between the knocker and the listener must flow can conduct its own attacks. Because this man-in-the-middle system can actually interfere with all the traffic, it could allow the knocking packet to flow as intended, but subsequently drop the knocker's connection attempt to the protected service. Obviously doing so can easily create a denial-of-service attack, but it also enables the man-in-the-middle to launch its own connection to the protected service.

By keeping the port out of the knocking packet's contents, a NAT time-of-request/time-of-use attack is mitigated. All implementations of Port Knocking are susceptible to man-in-the-middle attacks since the man-in-the-middle system can spoof the knocker's IP address, knowing that reply packets must be routed through it. In such a case, because all traffic must be routed through the man-in-the-middle, it is able to maintain connections without the interaction of the knocking system. Since the man-in-the-middle can also drop any packets it wishes, it can prevent the knocker and listener from being able to communicate, and neither party is aware that the man-in-the-middle attack is even occurring.

Like more traditional port-knocking schemes, single-packet authentication must rely on some form of covert channels to convey the port to be opened. The most secure means is through the port used in the knocking sequence. Using a pre-configured setting, both the knocker and the listener know which knocking port corresponds to which port to open. By allowing only a single connection per knock, the action is always to open the port, and upon connection or timeout threshold (whichever occurs first), the port is closed. Since this transcription method is easily detected by an eavesdropper, another covert channel is possible through the timestamp. The milliseconds of the timestamp can be used for a similar transcription scheme. Doing so would require a larger timeout threshold since latency cannot be calculated as accurately as when the milliseconds are not used in a transcription scheme.

IV. Proof of Concept

The basis of this work is derived from a blog post I wrote in February of 2013, where I outlined the possibility of utilizing asynchronous keys with Port Knocking for identification purposes [16]. In an attempt to test Port Knocking with single packet authentication for practicality I conducted a series of experiments. The experiments were mock-ups of a single packet knock consisting of a signature and sender's certificate. For the experiment, the knocking system and the listening system were geographically separated using the Internet cloud as the infrastructure. In addition, the knocker was behind a NAT device.

As mentioned previously, an attacker could exploit the vulnerability NAT creates and conduct their replay attack from within the same NAT'd network within the two second window. For this reason, my experiments used the IP address, TCP sequence numbers (both listed in the packet header), and timestamps as the plaintext message encrypted with the private key (thus creating a digital signature). Though the experimental system was never configured to this constraint, the intent is that only the first unique packet would be acknowledged, as the chances of the same sequence number being used in two connections originating from the same IP address with the same timestamp are astronomical. By doing so, replay attacks are thwarted.

As described earlier, the signature included in the packet's payload consists of the IP address, the TCP sequence number, and a UNIX timestamp including the milliseconds (as depicted in Figure 3).

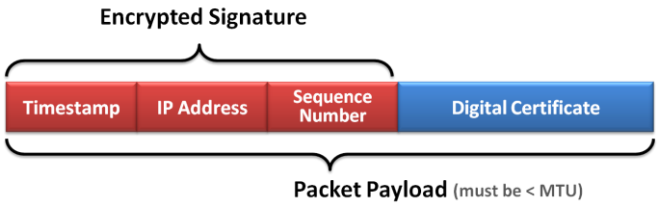

Figure 3. Single Packet Authentication packet contents.

The three data fields are then encrypted using the sender's private key, creating a signature. The remainder of the packet's payload consists of the sender's certificate (containing both their public key and the information about the Certificate Authority which issues the key pair).

A. Creation of Keys and Certificates

To remain in line with common practices on the internet, I decided on using the RSA asymmetric encryption algorithm. A script was created leveraging OpenSSL to generate 100 key pairs, along with a corresponding certificate and signature mockup. In the creation of the signatures, rather than generating them on-the-fly using the packet details (as specified above), the signature was created using mock data consisting of the UNIX timestamp in milliseconds and an ASCII string the same size as the IP address and sequence number. Upon experimentation, it was determined that there is no physical size difference for a signature using IPv4 data vice IPv6 data, as the size was the same in either case. An example of the script is seen in Figure 4.

```
#! /bin/bash
for n in {1..100}
do
  date +%s%N:abcdefgabcdefgh > plaintext$n.txt

  openssl genrsa -out private$n.key 1024

  openssl req -new -x509 -key private$n.key -out cert$n.crt -days 365 < input.txt

  openssl rsa -in private$n.key -pubout > public$n.key

  openssl rsautl -sign -in plaintext$n.txt -inkey private$n.key -out signature$n.pem

  c=$(openssl rsautl -verify -in signature$n.pem -inkey public$n.key -pubin)
  p=$(cat plaintext$n.txt)
  if [ $c != $p ]
  then
    echo -e "\e[00;31mError with key #$n!  Plaintext=$p Decyrpted Cyphertext=$c\e[00m"
  fi
  cat signature$n.pem cert$n.crt > payload$n.txt

done
```

Figure 4. Script generating RSA keys, certificates, signatures, and payload.

Line 6 of the script depicted in Figure 3 generates 1024 bit keys. Adjusting that line allowed for various experiments using different key sizes as the only variable. Line 8 uses a file named input.txt in order to respond to the certificate generation process, answering the questions such as location, country, name, and email address. The signature is created in line 12 and validated against the plaintext in line 18.

Modern practices for asymmetric key size are generally 1024 bits or more, with the current recommended size being 2048 bit. These experiments tested several key sizes to determine their impact on the signature size they each create as well as the digital certificate, as shown is Table 1.

Key Size	Signature	Certificate	Total Payload
1024 bits	128 bytes	1188 bytes	1316 bytes
2048 bits	256 bytes	1541 bytes	1797 bytes
3072 bits	384 bytes	1887 bytes	2271 bytes

Table 1. Single Packet Authentication signature contents.

To ensure the validity of the data in Table 2, keys were created on several systems at various times between February and October 2013. While 2048 and 3072 bit keys required actual values in the input.txt file, 1024 bit keys could be created with NULL values; however, it created certificates of 1379 bytes making the payload 1507. These certificates, unlikely in actual application, later proved to not be optimum in the experimentation.

B. Testing the Listener Trigger

In order to test the listener, I first needed the ability to send specially crafted packets to the listener. For this, I used Hping3—a packet generator and analyzer. Hping3 is a command-line open-source program which can create packets using a wide variety of options [14]. Pertinent to the experiments are the options to specify destination, packet count, port, packet size, and payload, as depicted below in Figure 5.

```
$ hping3 secureeves.com -S -c 1 -p 78 -E payload1.txt -d 1500
HPING secureeves.com (eth0 199.66.217.46): S set, 40 headers + 1500 data bytes
[main] memlockall(): Success
Warning: can't disable memory paging!

--- secureeves.com hping statistic ---
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Figure 5. Hping3 example syntax.

Like a traditional knocking packet, the packet depicted in Figure 4 was crafted as a TCP SYN packet, which would seem to be a connection attempt to any eavesdropper. The destination port was 78, using TCP as the default protocol. Only one packet was sent, consisting of the contents of the payload.txt file.

It is important to note that while one packet was sent, none were received. The reason there was no acknowledgement packet was because of a firewall rule set to block all ports. The next step was to validate that the packet was complete in its entirety, by capturing inbound traffic on the listener. A copy of the captured packet payload received by the listener appears as follows:

```
E....\..7...D2.w.B...0....rR"...P.......5...2D1..}.
...GhE.Y...*...t..1..>._._.g.7...,[.D...w.8.0..*.).
....EG.=d.N.{=p....>w..}.y..U..K...wp..q.....H.H.D.
.......k...l?.<-----BEGIN.CERTIFICATE-----.MIICtjCC
Ah+gAwIBAgIJAI0BLpsrpVBJMA0GCSqGSIb3DQEBBQUAMHQxCzA
JBgNV.BAYTAlVTMQswCQYDVQQIDAJJTDENMAsGA1UEBwwEVGVzd
DENMAsGA1UECgwEVGVz.dDENMAsGA1UECwwEVGVzdDENMAsGA1U
EAwwEVGVzdDEcMBoGCSqGSIb3DQEJARYN.dGVzdEB0ZXN0LmNvb
TAeFw0xMzEwMDYyMTE0MjNaFw0xNDEwMDYyMTE0MjNaMHQx.CzA
JBgNVBAYTAlVTMQswCQYDVQQIDAJJTDENMAsGA1UEBwwEVGVzdD
ENMAsGA1UE.CgwEVGVzdDENMAsGA1UECwwEVGVzdDENMAsGA1UE
AwwEVGVzdDEcMBoGCSqGSIb3.DQEJARYNdGVzdEB0ZXN0LmNvbT
CBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA.6AsT9VRsySJW
nlgwQrFF8ZUMxgXmqWgwDyVkSCtxrZ7ULIf2o55M5B/VnRE5LB4
g.1u0oHIqAbH/OcBW+9scBpm69m3nT7QutSlGLnWuP2OyAScRbl
D9z6ffCZC1GfbOS.CRcpLtGaREQnwkcfAXGrzgDNhZ2rzS2g8lL
dRr78MesCAwEAAaNQME4wHQYDVR0O.BBYEFFCN9I+4ppgjBQkIq
k5B/o/Yw8P+MB8GA1UdIwQYMBaAFFCN9I+4ppgjBQkI.qk5B/o/
Yw8P+MAwGA1UdEwQFMAMBAf8wDQYJKoZIhvcNAQEFBQADgYEAZP
S9OTB/.3XXXahLITLHfmMloQh8X40WzdkJof4ObLToub/Hl06+y
QECshPGKHJC2bZLd0tQp.xLkvK8sQEDJZNZij5PPO+6iYj571jT
fBrJH7ZpWtMzk3RtSyzZeH0uJEjHxZuWbz.S58IoLvlv84GxZfK
O2iyuMDCVsaxsUEpDGA=.-----END.CERTIFICATE-----.....
........................................................
........................................................
........................................................
........................................................
........................................................
.....................................
```

The captured packet contained the entire payload, with null value after the certificate (represented with a “.”). All data prior to the certificate is the signature, consisting of the encrypted timestamp, source IP, and sequence number placeholders. The signature and the certificate itself can easily be extracted from the packet using regular expressions parsing. Therefore, the packet contains all the information necessary for the listener to be triggered.

C. Findings

Though a success, the proof-of-concept experiments shed light to some hurdles which need to be overcome in implementation. Upon experimentation, it was discovered that 1500 bytes was the maximum packet size which was able to be transmitted, as it is the MTU of Ethernet (the Data-Link layer protocol for both the knocker and the listener). Though TCP’s MTU limit accommodates for any of the tested key sizes, truncation occurred on the packets larger than the Data-Link MTU. The only certificates which were small enough to fit inside the constraints of Ethernet were 1024 bit keys, which are quickly being replaced by the 2048 bit key standards in practical use.

It is possible to overcome the Ethernet MTU limit by including the signer’s public key rather than the certificate. Like the certificate, the public key can be easily parsed since it contains a similar preceding and tailing identifier string. Most importantly, the public key for a 3072 bit key can easily fit in the MTU of Ethernet. The downfall for this approach is quite obvious: trust is established by the digital certificate and it’s CA. However, since a port-knocking deployment is intended for members of a selected group, they would all pull from the same CA.

By using an in-house CA, the listener would be able to cross-reference the public key with keys that have been issued by the CA. Since it is highly unlikely that two members have been issued the same public key, the certificate will likely be found. However, querying the CA for key information may be too time-consuming for Port Knocking uses. Alternatively, the listener can be configured with the certificates of its knockers locally, resulting in faster cross-referencing. These approaches becomes more flawed as the number of created keys increases since the likelihood of two certificates using the same public key also increases. Therefore, it is recommended to only use this approach for smaller networks which require fewer keys to be generated and thus cross-referenced.

The experiments also proved that the standard firewall log levels cannot be used to extract the contents of a packet. Two possible approaches compensate for this issue. The first approach would be to modify the firewall logs to include the payload of blocked traffic. Doing so, however, could cause substantially larger logs generated during a port scan or similar network attacks occurs. It is possible to streamline the logging process, minimizing the amount of denied traffic while still maintaining a longer history of potentially more relevant logs.

Another approach is to build the listener into the firewall itself. By building the listener into the firewall, the time delay between packet reception, analysis, and rule-modification is optimized. However, this would in-turn provide a mechanism for the firewall to modify its own rule, which potentially create other security issues.

V. Future Work

Where there are several Port Knocking suites available, this experiment utilized shell scripts rather actual applications. It is quite feasible to either incorporate this implementation into an existing Port Knocking suite or develop a solution from scratch [5][8]. A live program operating in memory is sure to prove more responsive than simple scripts.

The limiting factor in the experiment was the use of Ethernet as the Data-Link protocol. Other protocols exist which allow for a higher MTU; but, currently Ethernet is the de facto Data-Link protocol. As technology changes, it is possible that another protocol will become common; one which allows for a larger MTU.

As mentioned previously, RSA was used in this experiment because of its widespread use on the Internet. Elliptic Curve Cryptography (ECC) is an asymmetric algorithm which has significantly smaller certificate [16]. It is possible that much stronger ECC keys may be used in lieu of RSA while still fitting within the Ethernet MTU limitation.

Though the proof-of-concept test used TCP, it is possible to implement Port Knocking with single packet authentication relying upon any IP-based protocol. UDP, for instance, can conserve 12 bytes for additional payload content; however, the use of sequence number is a TCP-specific implementation.

References

[1] M. Krzywinnski. “Port Knocking — Network Authentication Across Closed Ports”, *Sys Admin*, XII (6), pp. 12-17, 2003.

[2] M. Krzywinski. “Port Knocking”, *Linux Journal*, pp 1-3, 2003. [online]

[3] D. Isabel. “Port Knocking: Beyond the Basics”, *SANS Institute InfoSec Reading Room*, SANS Institute, 2005.

[4] C. Pfleeger, S. Pfleeger. *Security in Computing*, Prentice Hall, Upper Saddle River, pp. 151-160, 2006.

[5] M. Krzywinski. “Port Knocking from the Inside Out”, *PortKnocking.org*, 2005. [online]

[6] S. Miklosovic. “Port Knocking Enhancements”. Masaryk University, Kamenice, Czech Republic, 2011.

[7] R. DeGraaf, J. Aycock, M. Jacobson Jr. “Improved Port Knocking with Strong Authentication”. University of Calgary, Calgary, Canada, 2005.

[8] M. Rash. “Single Packet Authorization with FwKnop”. *Cyberdyne.org*, 2005. [online]

[9] S. Jeanquier. “An Analysis of Port Knocking and Single Packet Authorization”. *Royal Holloway College*, University of London, London, United Kingdom, 2006.

[10] B. Schneier. *Applied Cryptography*, John Wiley & Sons, Inc., New York, 1996.

[11] E. Maiwald. *Network Security: A Beginner’s Guide*, McGraw Hill, New York, pp. 253-265, 2013.

[12] S. Anson, S. Bunting. *Mastering Windows Network Forensics and Investigation*, Wiley Publishing, Inc., Indianapolis, pp.79-87, 2007.

[13] J. Kim, R. Phan. “Advanced Differential-style Cryptanalysis of the NSA’s Skipjack Block Cipher”, *Cryptologia*, XXXIII (3), pp. 246-270, 2009.

[14] S. Davidoff, J. Ham. *Network Forensics*, Prentice Hall, Upper Saddle River, pp.430-438, 2012.

[15] T. Simonite. “Math Advances Raise the Prospect of an Internet Security Crisis”, *MIT Technology Review*, pp. 1, 2013. [online] Available from: http://www.technologyreview.com/news/517781/math-advances-raise-the-prospect-of-an-internet-security-crisis/

[16] M. Reeves. “Knock, Knock…”, *Secureeves.com*, pp. 1, 2013. [online] Available from: http://www.secureeves.com/blog.php?id=40

[17] T. Rosati, G. Zaverucha. “Elliptic Curve Certificates and Signatures for NFC Signature Records”, *Research in Motion, Certicom Research*, pp. 10, 2011.

Author Biography

Michael Reeves Born in Greenbay, Wisconsin, Michael has worked professionally in the Computer Security field for over 10 years. In 2011, he earned a Master’s of Science degree in Information Technology (with a specialization in Information Assurance) from the University of Maryland University College, located at Adelphi, Maryland. Previously, in 2008 Michael received a Bachelor’s of Science in Management of Computer Information Systems from Park University, located at Parkville, Missouri. In addition he holds CISSP, A+, Network+, Security+, and Linux+ certifications.