UNIVERSITY OF SOUTH FLORIDA

Port Knocking Network Protocol

John Cameron and Jordano Baer

November 28th, 2018

1 SERVICE TO BE PROVIDED

The service to be provided is a hidden (or invisible) web server that is visible only after a valid port knocking protocol has been completed by a client.

2 ASSUMPTIONS ABOUT THE ENVIRONMENT

The environment is the Internet where the web server is assumed to be running on a reachable (so, fixed IP address) host. The host supports the full set of Internet protocols. In order for the protocol to function correctly, the server and client must both have the same unix time with minute accuracy and a propagation delay of less than a minute.

3 VOCABULARY OF MESSAGES

KNOCK (*server-bound packet*)

The only message in this protocol is a Knock. Notice that there is no packet sent by the server to the client whether the knock sequence is correct or not. The reason for this decision was to maximize security by reducing noise and preventing potential eavesdroppers from knowing if a knock was correct or not.

4 ENCODING OF MESSAGES

Each packet is transported using UDP because of its light overhead, support for multiplexing, and connection-less property. All packet fields are in Big Endian. For this protocol, 1 byte is equivalent to 8 bits. Figure 4.1 is a diagram for the encoded payload of a Knock packet.

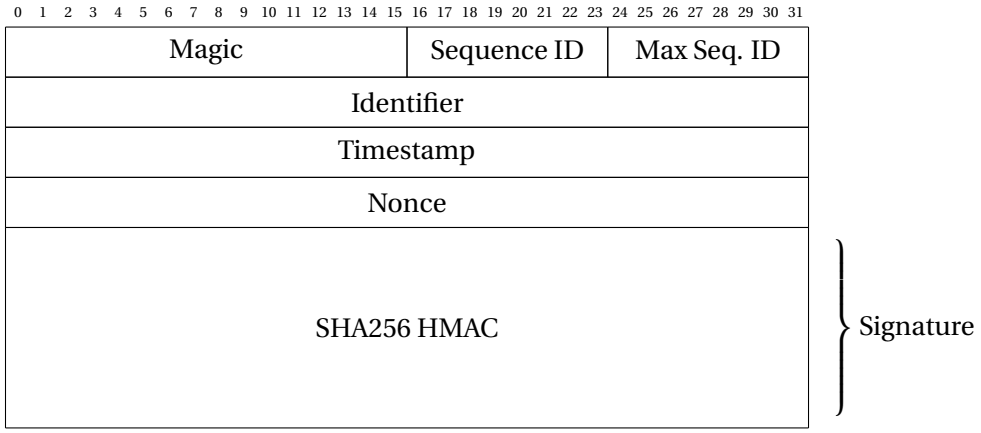


Figure 4.1: Encoded UDP Knock Packet

MAGIC Constant string of 2 bytes used to confirm that the payload is part of the Port Knocking protocol. The magic field should always be the ASCII encoded value of PK (an acronym representing the phrase "port knocking").

SEQUENCE ID A 1-byte signed integer. Each knock packet received by the server will be processed in ascending order based on the sequence ID.

MAXIMUM SEQUENCE ID A 1-byte signed integer. The maximum sequence ID (i.e. length) of the knock sequence being transmitted from the client to the server.

IDENTIFIER The identifier is a 4-byte ASCII encoded string ID used to identify the key used to sign the packet payload.

TIMESTAMP The timestamp is a 32-bit unsigned unix epoch time integer of when the packet was transmitted.

NONCE The nonce is an incrementing, consecutive unsigned 32-bit integer used for countering replay attacks.

SHA256 HMAC A 32-byte message authentication code (MAC) with the shared secret identified by the identifier field. Used for data integrity and authenticity of the packet. All fields before the MAC are considered in the calculation of the MAC.

5 PROCEDURE RULES

5.1 GENERAL CONCEPT

SERVER While the server is running, it will listen on a set of n port numbers for incoming Knock packets from a client. Every minute, the server will change its set of port numbers using a pseudo-random number generator. In a pseudo-random generator, it is possible to predict generated values if the seed is known. This concept is critical in order for the server and client to know what port numbers to communicate on. Once the server receives a Knock packet on any port, it will decode the packet,

verify client authenticity, verify the data integrity, and then record a new knock from a client identified by the $identifier$ field (See Figure 4.1). After all n knock packets have been received by the server (one for each port), the server will either open the hidden web server if the knock sequence (port numbers that each packet was received on) is correct or do nothing. The order of what ports each Knock packet was received on is known as a knock sequence.

CLIENT The client will first generate a sequence of n port numbers by using a pseudo-random number generator that utilizes the same seed in server. Then, the client sends a Knock packet to each port number in the generated sequence to the server. Because the server does not reply with a status packet, the client needs to assume that the hidden service is now open. If the service is not open, the client should then start from the beginning. After three attempts, the client should give up and assume the server is offline. For the remainder of the procedure rules, three knock packets are used as an example. Although, it is important to note that it is actually n knock packets (one for each listening port).

5.2 SHARED SECRET

There are two shared secrets each client must know in order to successfully transmit an accepted knock sequence: the port secret and client secret. The port secret is static, and is used in a pseudo-random number generator to determine what the destination port should be for each knock packet. The client secret, however, is unique for each client and is used in the MAC. The client secret is also associated with an identifier so that the server can identify the shared secret used to create the Knock packet.

5.3 GENERATING PSEUDO-RANDOM UDP PORTS

The ports used for a knock session are generated by a pseudo-random number generator seeded with a hash of the port secret and current unix time in minutes. Algorithm 1 is pseudo-code that describes the process of generating three unique ports that only the server and client will know based on the port secret. The range of a possible generated port is [1025, 65535]. The minimum bound was set due to most operating systems having ports 0-1024 reserved for standard applications (such as HTTP on port 80). The maximum port is equivalent to the maximum port value permitted by UDP.

Algorithm 1: Pseudo-random port number generator

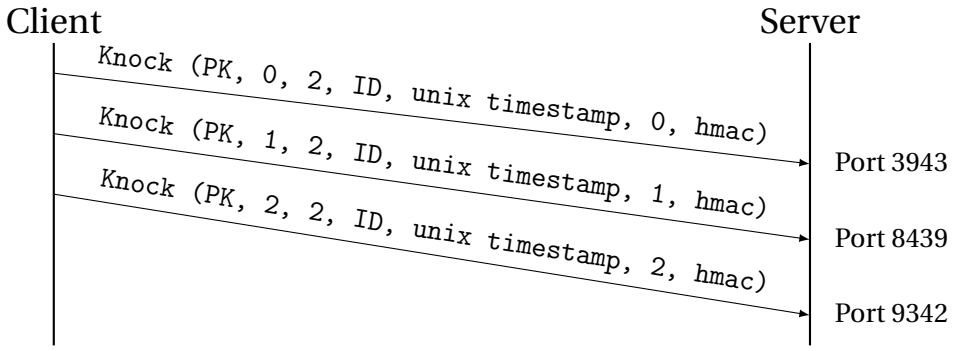
Input: A port secret and the number of ports to generate
Output: n distinct port numbers

```
1  Function generatePorts(portSecret, n):
2      time = (current unix timestamp) / 60
3      hash = md5(portSecret + time)
4      seed random generator with hash
5      ports =Array(n)
6      while ports.size() ≠ n do
7          port = random integer within [1025, 65535]
8          if port ∉ ports then
9              ports.add(port)
10         end
11     end
12     return ports
```

5.4 Normal Knock Sequence

Figure 5.1 is a timing diagram that illustrates a successful knock session between a server and client. The knock sequence is a series of pipelined Knock packets sent from the client to the server. In the diagram, the client is sending three UDP Knock packets on the generated port numbers 3943, 8439, and 9342.

Figure 5.1: Normal Knock Sequence Timing Diagram



Notice that there is never a packet sent from the server during the knock sequence. This helps avoid generating extra noise within the network and also ensures that an eavesdropper will not know of whether the sequence was correct or not. This, however, comes at a cost of the client not knowing if the sequence was received by the server. As a result, the client should assume that the server always received the sequence. If the service is unavailable, the client should send a new knock sequence. After a certain number of attempts, the client should then assume the server is offline.

5.5 Client Authentication

Because packets are not encrypted in this protocol, anyone is able to view the payload in clear-text. Because of this, it is important to implement a measure to counter spoofed and modified packets. The solution is to require each packet sent to the server contain a key-hashed MAC appended to the end of the payload (known as the signature in Figure 4.1). The shared secret (key) used for calculating the MAC is determined by the identifier. Each client will have a unique shared secret and its associated identifier (similar to the common name in a certificate) which the server can then use to verify the authenticity and integrity of the payload. If the server receives a MAC that is corrupt or invalid, then the packet must be dropped. Algorithm 2 is pseudo-code that describes the algorithm on the server that verifies the authenticity and integrity of a received packet. In the algorithm, the map is an associative map with a key of the client identifier and a value of its assigned shared secret.

Algorithm 2: Message authentication code verification

Input: The client identifier, packet payload (all fields before the signature), and the received MAC
Output: A Boolean indicating if the MAC is either valid or invalid

```
1  Function verifyMAC(identifier, payload, mac):
2      sharedSecret = map.Get(identifier)
3      if sharedSecret = NIL then
4          return False
5      end
6      mac_algorithm =A HmacSHA256 algorithm
7      Initialize mac algorithm with the shared secret
8      real_mac = mac_algorithm.doFinal(payload)
9      if real_mac = mac then
10         return True
11     else
12         return False
13     end
```

5.6 Sequence IDs

Sequence IDs are included in the Knock packet in order for the server to determine the order of the knock. This is because the transport layer protocol, UDP, does not guarantee the order of incoming packets. The sequence IDs are consecutive integers, with the maximum number being defined in the max sequence ID field. An additional benefit of using sequence IDs is that the server will be able to determine once the client is finished sending data (similar to a TCP segment with the FIN flag set). Figure 5.2 demonstrates how sequence IDs are used to determine the order of the knock packets. Notice that the second packet was transmitted through the internet faster than the other two packets. As a result, the second packet got received by the server out-of-order.

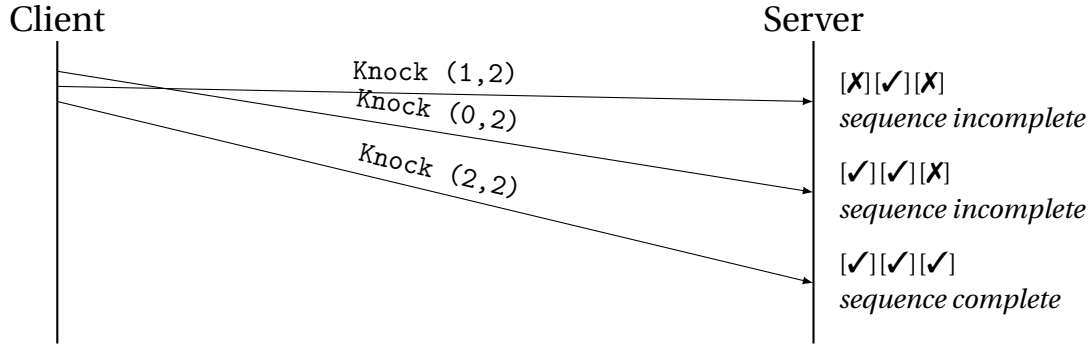


Figure 5.2: Timing Diagram showcasing the purpose of Sequence IDs

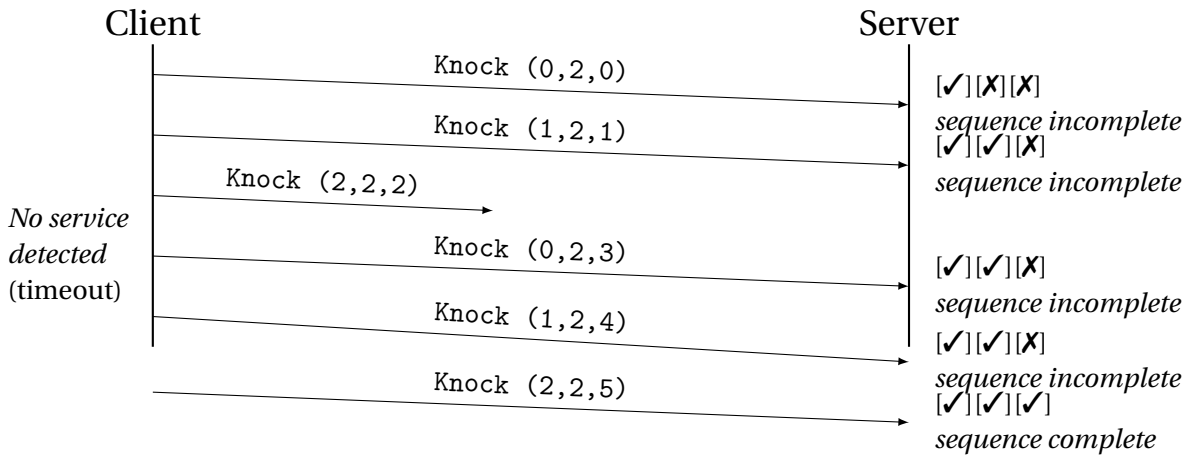
5.7 Timestamp and Nonce

The fields used to counter replay attacks are the timestamp and nonce. Both fields are for an unsigned 32-bit integer. There is an assumption that the clock on both the server and client is correct. In other words, the server and client are able to have the same unix time with minute accuracy. The server will drop any packets with a time difference of over one minute. Any packet received with a timestamp less than the current unix time one minute ago must be dropped. The value of the nonce is predetermined by the client according to its configuration. Every time that a packet is transmitted, the nonce will be incremented by 1 on the client-side and re-saved in the configuration for future reference. For the server, the nonce will be recorded whenever a verified packet has a nonce greater than what is logged on the server.

5.8 Packet Loss

There are three possible cases that can result in a failed knock sequence: out-of-order, corrupt data, and packet loss. The first two, out-of-order and data corruption, already have solutions designed (Sequence IDs and Data integrity verification). However, because the transport layer protocol used is unreliable, knock packets could be lost. The client will implicitly determine if packet loss occurred if the hidden service hosted on the server is not available. In this case, the client should send a new knock sequence. Figure 5.3 demonstrates the situation of when packet loss occurs. The first integer for each Knock packet is the sequence ID, the second integer is the maximum sequence ID, and the third integer is the nonce.

Figure 5.3: Packet Loss Timing Diagram



5.9 Control Flow Diagrams

Flowcharts were decided to illustrate the behavior of the control flow for the server and client because of the stateless decision for this protocol. The protocol was designed to be stateless in order to support multiple clients and parallelism. Each flow chart represents the control flow in a thread.

5.9.1 Server

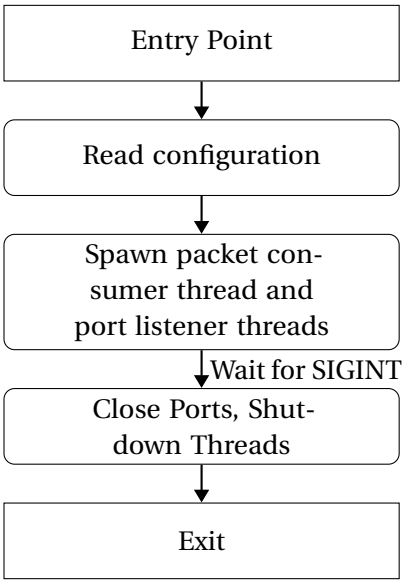


Figure 5.4: Main Thread Flowchart for Server

Figure 5.5: Packet Listener Thread Flowchart for Server

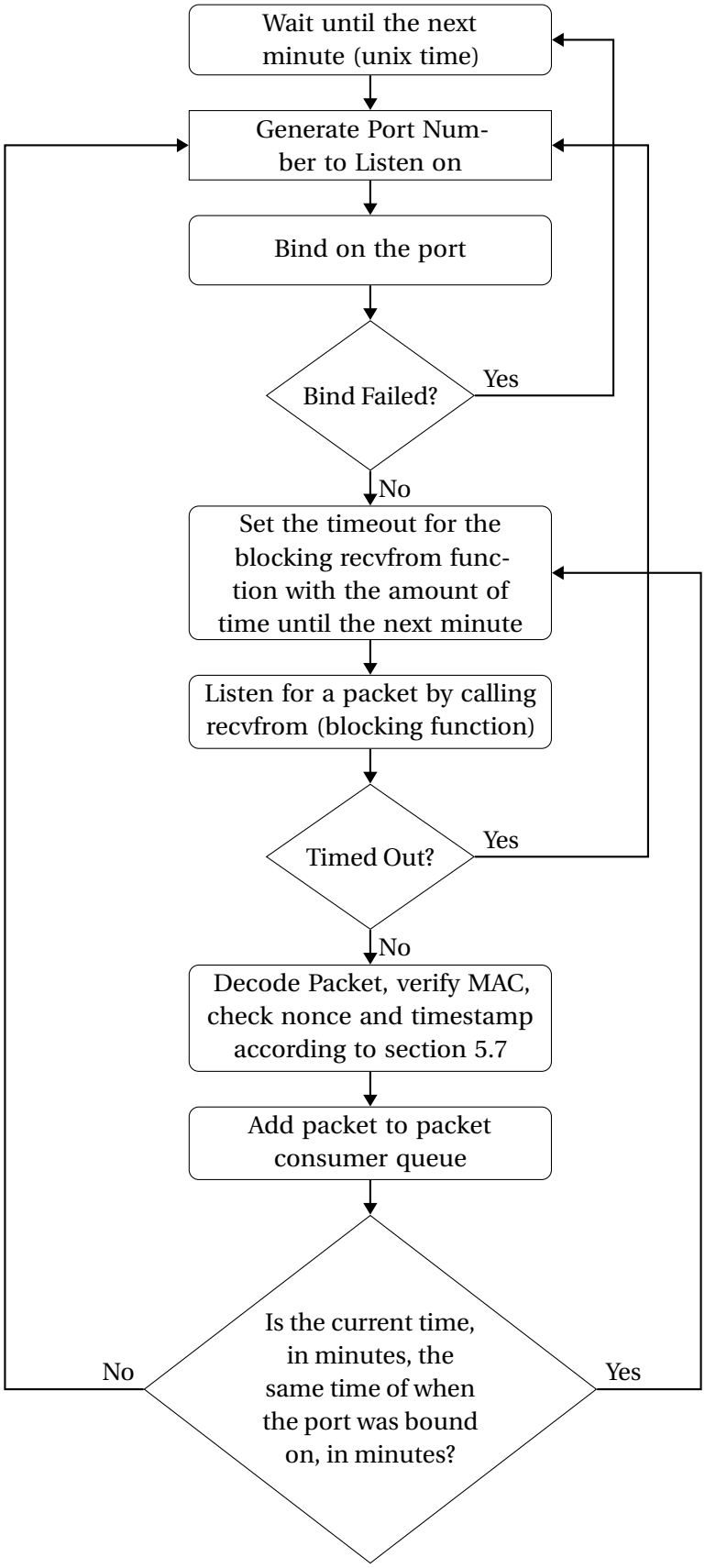
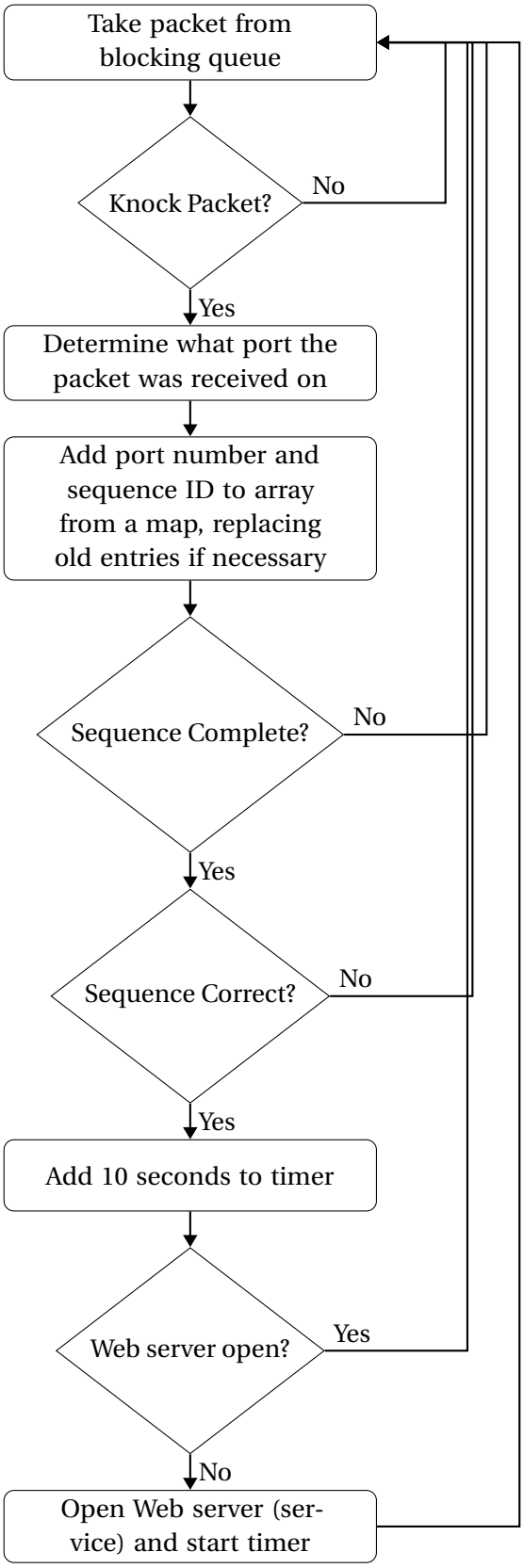


Figure 5.6: Packet Consumer Thread Flowchart for Server



5.9.2 Client Flowchart

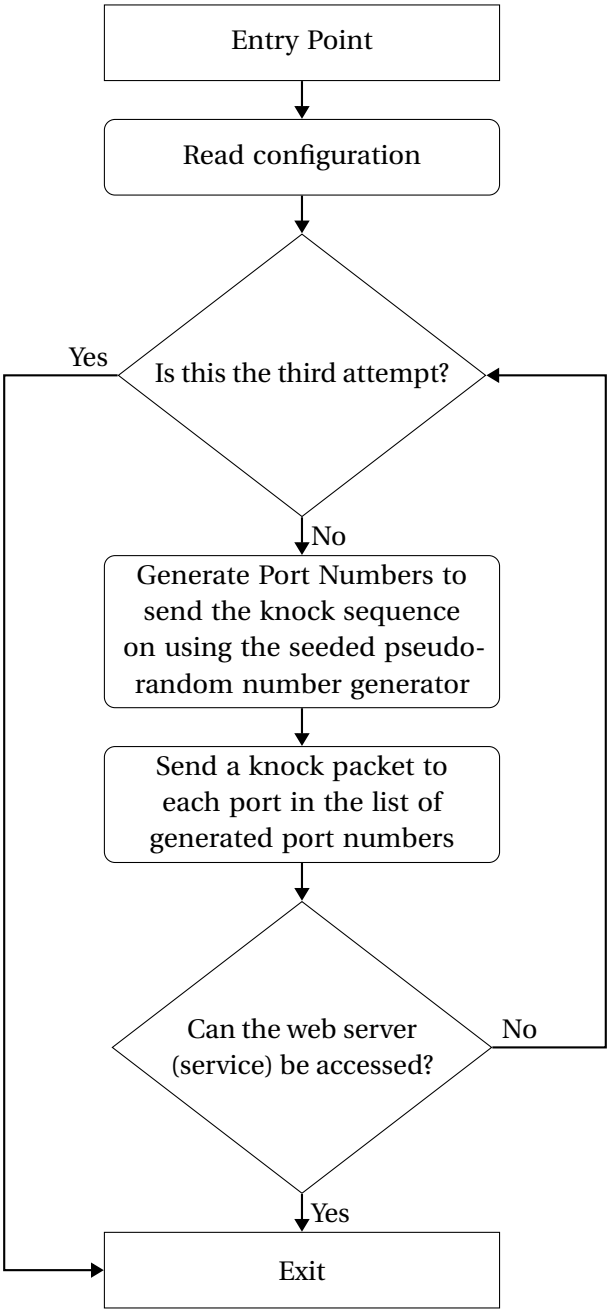


Figure 5.7: Flowchart for Client

6 Implementation

It was decided that Java 8 would be used as the programming language of choice for implementing this protocol because it is an object-oriented and concurrent language, which allows multiple users to be supported simultaneously with ease. Additionally, Java also includes a majority of necessary algorithms, functions, data structures, and many third party libraries if needed. For example, the

SHA256 HMAC algorithm is included in the Java Cryptography API.

6.1 Program Structure

Gradle, an open-source build automation system, was used for the program structure because the project applies the concept of modular programming. Additionally, basic unit testing procedures are executed after the compilation stage to ensure each module is correctly programmed.

6.2 Configuration

The configuration file, which is a Java properties file, allows an administrator to configure the necessary settings for the server and clients. Below are the default settings and values for each configuration file.

```
# Information to locate the server
server-address=localhost
port-secret=portSecret
ports=3
# Information used for authentication
client-identifier=com1
client-shared-secret=com1_secret
nonce=0
```

Listing 1: Default client.properties

```
# The bind address. Can either be an IP Address or Host Name
bind-address=0.0.0.0
# The secret used along with the current time to determine what ports to listen on for incoming
    knock packets
port-secret=portSecret
ports=3
trusted-clients-path=trusted_clients.txt

# The timeout, in seconds, before closing the service
open-timeout=10
```

Listing 2: Default server.properties

```
# Set to true if the embedded web server should be used
use-embedded-webserver=true
# The address to bind the network service on
# (could be the embedded web server or some other service)
service-bind=127.0.0.1
service-port=8080
# If not using an embedded web server, a transparent TCP proxy will
# be used for forwarding requests to the other web server.
# The address settings for the remote host
proxy-address=
proxy-port=
```

Listing 3: Default service.properties

```
# The identifier, shared secret, and nonce are delimited by a space
com1 com1_secret 0
com2 com2_secret 14
com3 com3_secret 5
```

Listing 4: Example of trusted_clients.txt

6.3 Modules

The idea behind the source code was to divide the client, server, web server, and protocol into separate modules. These modules are permitted to be dependent on one another if it is necessary for another module. For example, the Server module is dependent on the Protocol and Service modules.

Server Module Implements the procedure rules for the server.

Client Module Implements the procedure rules for the client.

Protocol Module Implements the protocol encoding scheme and vocabulary.

Service Module Implements a standalone service. In this implementation, the two services are an embedded web server capable of serving requests required by the test cases and a transport layer proxy.

6.4 Simultaneous Connections

The server has been designed to scale effectively to support multiple clients and connections. In order to support multiple clients, it is important that the processing delay between accepting a packet and waiting for the next one is minimized. For this to happen, each bounded port accepted an incoming packet on its own thread. In addition, another thread is employed for decoding packets to further minimize the processing delay. When a knock sequence is being received by the server, it is recorded in a key-value pair map, with the key being a client identifier (configured by the server administrator) and the value being an array of recorded Knock packets.

6.5 Using another web server

By default, this implementation will spawn its own embedded web server. The embedded web server used is NanoHTTPD. If the administrator wants to use another web server, then the implementation will start and kill the external web server process. At first, a reverse HTTP/TCP proxy was considered in order to cleanly control access to the web server. The proxy would forward requests to the web server if it is considered open and drop requests if it is considered closed. To prevent a user from bypassing the proxy, the webserver would bind to the local loopback address of the machine (or some other non-public interface). However, this idea fails when HTTPS is employed because the client will terminate the connection with the proxy since the common name in the SSL certificate sent by the remote server does not match the proxy address. But because the web server is only required to be able to serve basic HTTP GET requests according to the project requirements, this idea was implemented.

7 Vulnerabilities

7.1 Spoofing and Data Tampering

Spoofing and data tampering is not possible in this protocol because each packet must include a keyed-hash message authentication code. Any received packet must verify this MAC, which checks for data authenticity and integrity of the packet.

7.2 Playback Attack

The protocol is immune to playback attacks because of the nonce and timestamp. The nonce is a consecutive and unique integer for each knock packet. Additionally, the server and client both log the last nonce value sent/received. At first, it may seem unnecessary to include the timestamp if there is a nonce available. The timestamp is included in order to mitigate a very specific situation for which can attacker can perform. Suppose an attacker is doing an active man-in-the-middle (MITM) attack. The attacker could read incoming knock packets, remove them from the wire, and save the byte stream for a future transmission. The timestamp field is a counter-measure for this type of attack.

7.3 UDP Flooding

The procedure rules designed for this protocol have an intriguing property to help counter UDP flooding and computational resource exhaustion. Because the port numbers are pseudo-randomly generated based on a seed of the unix time in minutes and the port secret, an attacker would have a difficult time determining what ports to flood in order to overload the application. This is further enhanced by the server never responding to any knock packets.

7.4 Resource Exhaustion

Knock Sessions Duplicate knock packets replace current knock packets in the saved session per client. The memory allocated for knock packets is the amount of knock packets expected by the ports configuration setting. It is not possible for the client to cause the server to allocate for memory than required (e.g. sending a packet with a maximum sequence ID of 1000+).

Computational Exploit An attacker is able to exploit the computational resources required for cryptography procedures (e.g. HMAC calculations) by flooding the server with properly encoded Knock packets. In the server implementation, there is an buffer queue for incoming UDP packets that need to be decoded. If the attacker is able to fill the buffer (size determined by the server administrator) faster than the server can process them, then any new incoming packets will be dropped due to buffer overflow. A solution to help resist this type of attack is by using multiple threads to help distribute the work load on the server. Another solution implemented is the randomization of the ports.

7.5 possibility of Web server being attacked

The web server is still able to be attacked if the web server is open. An attacker could simply ping the web server to see if the service is available. If it is, then the web server can be attacked with DoS by flooding packets on the port bound by the web server. A solution to this problem would be to use the pseudo-random port number generator algorithm to generate a port that the web server would listen for while the service is open. After the web server port is closed, a new port will be assigned for when the web server will be open again.