Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bachelor Thesis

Network Security Group, Department of Computer Science, ETH Zurich

Data Plane Security Aspects in Next-Generation Internet Architecture Design

by Dominik Lehmann

Summer 2021

ETH student ID:	18-916-320
E-mail address:	lehmando@student.ethz.ch
Supervisors:	Joel Wanner Prof. Dr. Adrian Perrig

Date of submission: September 26, 2021

Abstract

The Internet has continued to suffer from attacks on its data plane for multiple decades. Distributed denial-of-service (DDoS) attacks remain a significant threat to today's Internet ecosystem, despite efforts to develop secure protocols. One reason for the slow progress in this area can be attributed to fundamental design principles of the BGP/IP protocol stack. SCION, a future Internet architecture, chooses a radically different approach. The SCION [8] architecture provides multi-path communication and departs from classical allocation of address prefixes to autonomous systems (ASes) by addressing hosts using a multi-level scheme.

The goal of this thesis is to investigate the fundamental implications of such Internet architecture design choices. We achieve this by developing concrete data-plane attacks and evaluating their effect using the SCIONLab testbed, which enables experimentation with a future Internet architecture. The first attack is a more refined version of a DDoS attack, which exploits SCIONs multi-path communication. The second attack prepends imaginary ASes to a legitimate path to create a spoofing attack against a server by creating many QUIC sessions in parallel, thus depleting the server's resources.

Acknowledgements

I want to thank Prof. Dr. Adrian Perrig for the opportunity to complete my Bachelor's Thesis in the Network Security Group.

I also wish to extend my special thanks to Joel Wanner for his supervision and guidance throughout this project. His insight into SCION helped me out quite a few times.

Lastly, I would like to thank the SCION developers team, who were always quick to respond to problems of all sorts.

Contents

1 Introduction

The Internet has continued to suffer from attacks on its data plane for multiple decades. Despite collective efforts to develop secure protocols, mitigation techniques, and establish best practices for routing security, distributed denial-of-service (DDoS) attacks remain a significant threat to today's Internet ecosystem. Reasons for the slow progress toward eliminating these threats can be attributed to fundamental design principles of the BGP/IP protocol stack: source address spoofing is still possible in many circumstances, which enables attackers to conceal their identity.

Proposals for future Internet architectures often choose radically different approaches. For instance, the SCION architecture provides multi-path communication and packet-carried forwarding state (PCFS), where each packet contains cryptographically protected information of its AS-level path. Moreover, the design departs from the classical allocation of address prefixes to autonomous systems (ASes) by addressing hosts using a multi-level scheme.

Such design choices have wide-ranging implications for the security, reliability, and performance of the resulting architecture. While multi-path communication enables much more reliable communication between two endpoints, as the path can be switched immediately in case of congestion or an attack, there may be adverse consequences to such an architecture. *Temporal Lensing* [9] attacks, a type of sophisticated DDoS attack, could cause much damage if the attacker has access to a multi-path communication system. The first part of this thesis deals with the implementation of a *Temporal Lensing* attack in the SCION network and measuring its efficacy.

Furthermore, the division of the Internet into multiple ASes may, contrary to SCION's intention, allow for even more malignant spoofing attacks, where the attacker, in addition to the source IP address, also spoofs different source AS addresses. The second part of this thesis deals with implementing a spoofing attack, which establishes many QUIC sessions with a server in parallel.

1.1 Contributions

- Implementation of a temporal lensing attack utilizing the multi-path communication of SCION

- Implementation of a two-way spoofing attack by path extension, which establishes many QUIC sessions with a server in parallel

- Measurements of the attacks as mentioned above
- Proposals for mitigation of the spoofing and temporal lensing attack

2 Background

2.1 SCION

SCION is a next-generation path-aware Internet architecture and alternative to the much-used BGP/IP model, which is predominantly used nowadays. SCION is centered around autonomous systems (AS) and isolation domains (ISDs). Each node in Figure 2.1 represents an AS, and several ASes are grouped into one ISD. An ISD is administered by several ASes, called core-ASes which negotiate a policy called trust root configuration (TRC). The TRC defines roots of trust to validate bindings between public keys and identities. Intra AS communication is mostly the same as in today's Internet, where an IP address is enough to send a packet to its destination address. It is important to emphasize that ASes do not share an IP address space, meaning that a particular IP can appear in multiple ASes. The big difference of SCION compared to today's Internet architecture is noticeable in the inter-AS and inter-ISD communication. Additionally to the destination address, a path is needed to route the packet to its destination. Abstractly speaking, a path is just a concatenation of ASes from the source AS to the destination AS. The addition of a path allows the sender some control over how the packet is forwarded to its

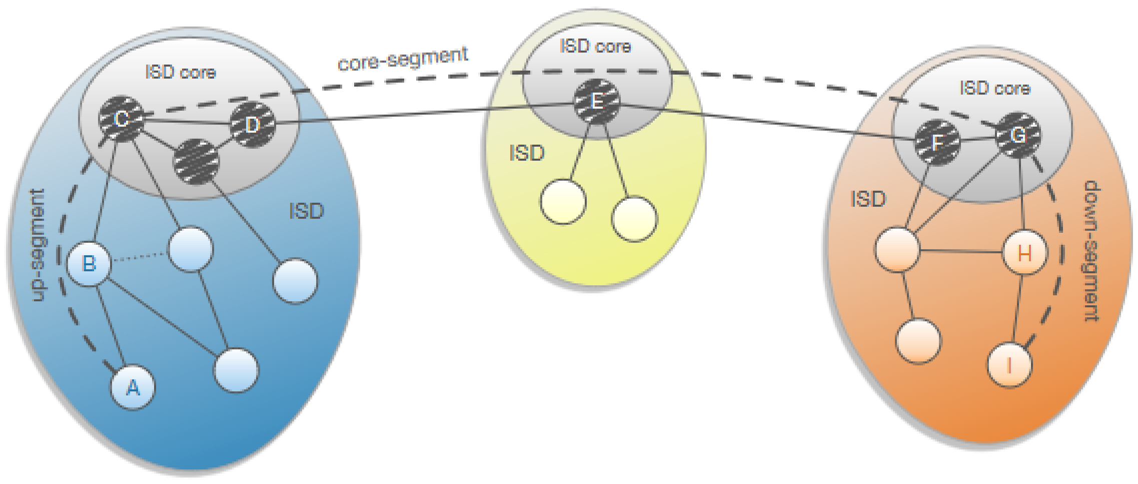


Figure 2.1: A SCION network topology. From Perrig et al. [8]

destination. The existence of multiple paths also increases the availability in the presence of adversaries. As long as a path exists, where the adversary does not have any control, this path can be used to communicate between two endpoints. Furthermore, if there is congestion on a path or even loss of connectivity, the sender can quickly evade to another path.

2.1.1 Path Construction

The creation of a path in SCION starts when the ISD core sends out path-segment construction beacons (PCBs) to other ASes within its ISD, which happens periodically. As the PCBs traverse the network, they accumulate cryptographically protected AS-level information like ingress and egress interface, peering interfaces, and the AS name of each traversed AS. All this information about an AS is called a Hopfield. By passing through multiple ASes the PCB chains the resulting Hopfields together into a path-segment, which is then stored at each AS's beacon server (Fig. 2.2). Core PCBs work the same way, except they establish path-segments between Core ASes, optionally between core ASes of different ISDs.

To construct a path between different ASes, we need up to three segments. In Figure 2.1 we see how an inter ISD path between AS A and AS I is constructed. The first segment is mandatory and is the *up-segment* (A to C in Fig. 2.1). It is the

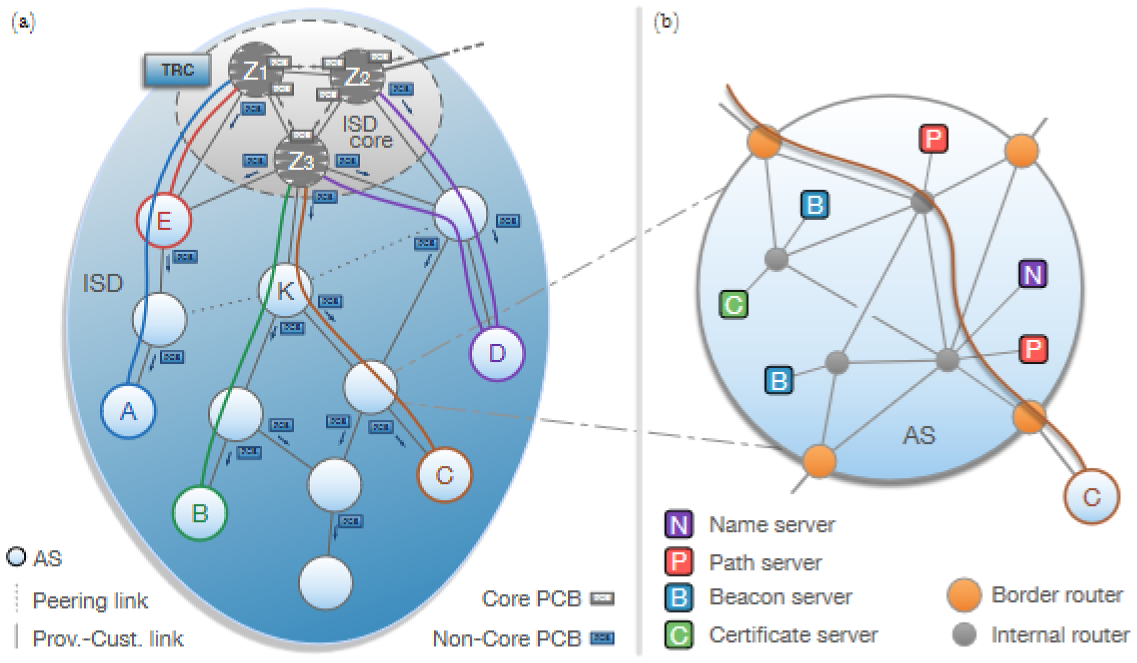


Figure 2.2: A SCION network topology. From Perrig et al. [8]

segment from the Hosts AS up to the ISD core. This segment can be created by flipping the order of ASes from the PCB path-segment that traveled from the core AS C down to A. Next is the *core-segment*, which creates a connection between two ISDs (core AS C to core AS G in Fig. 2.1. Lastly, the *down-segment* describes the path from the AS core to a non-core AS in the same ISD (G to I in Fig. 2.1). There can also be paths that are constructed with only one segment. If, for example, a Host in AS A wanted to communicate with a Host in AS B, only the up-segment would be needed (Fig. 2.1).

2.1.2 Packet Headers

In the following, we will see how the information about a path is encoded inside a packet traveling through the SCION network. This encoding is called a packet-carried forwarding state (PCFS). The name comes from the fact that forwarding at border routers occurs according to the path information a packet carries, compared to the conventional Internet, where forwarding happens based on IP forwarding tables.

(Path Type: SCION) In Figure 2.3 we see the layout of the bytes inside a SCION packet that encode the path. First is a PathMetaHdr, a 4-byte header containing meta-information about the SCION path. Next, there can be up to three InfoFields. Each InfoField has meta-information about a segment of the path: (1) up-segment, (2) core-segment, (3) down-segment. In the case of intra-AS communication, there can also be zero InfoFields. Following the InfoFields, there can be up to 64 HopFields. HopFields correspond to the different ASes the packet passes through on its path to its destination.

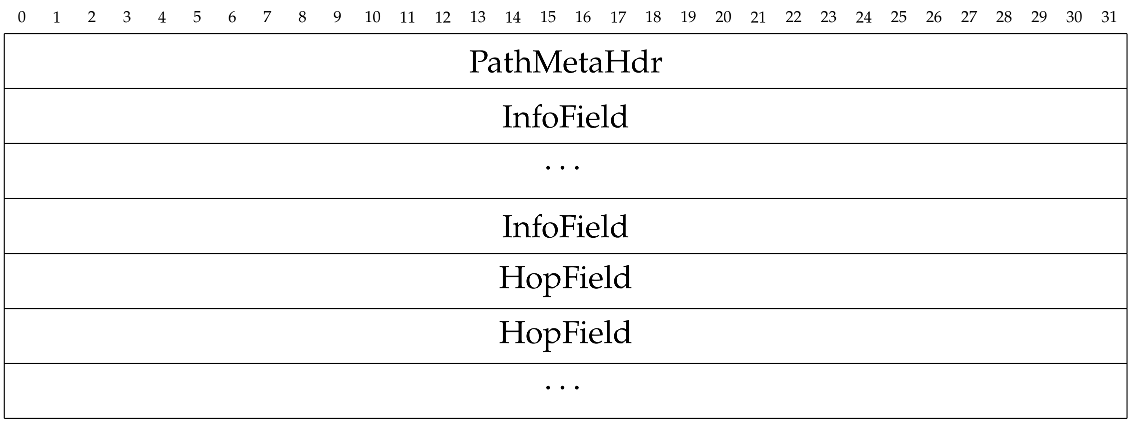


Figure 2.3: Path Type: SCION [3]

(PathMeta Header) The PathMeta Header's (Fig. 2.4) primary function is to track where the packet is during transit. It points to the Hopfield (AS) the packet is currently inside and which segment this Hopfield belongs to (C points to the current segment and CurrHF to the current Hopfield). Furthermore, the Meta Header contains information about the various segment lengths, length meaning how many Hopfields each Segment contains. If a segment has length zero, the path does not use that segment, and the corresponding InfoField is skipped.

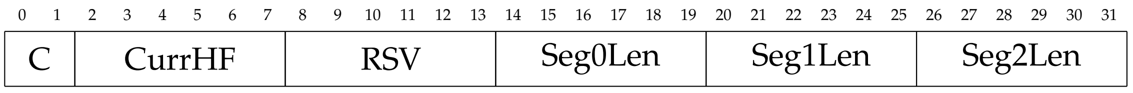


Figure 2.4: PathMeta Header [3]

(Info Field) The main task of the InfoField (Fig. 2.5) is to validate paths. When a PCB is created and sent out from a core AS, a Timestamp is set, and a random SegID β_0 is chosen. Every time the PCB enters another AS the current SegID β_i is updated by the AS, which XORs its MAC σ_i to the current SegID β_i returning a new SegID β_{i+1}. This MAC-chaining allows border routers to remove packets that have invalid SegIDs, which prevents Hosts from reordering Hopfields or adding additional Hopfields to the path (there are cases where this is still possible, which we will see later).

Border routers check the validity of a path inside a packet by comparing Hopfield MACs to the SegID. Whenever a packet arrives at a border router during AS traversal, the border router reverses the MAC-chaining by XORing the MAC σ'_i of the current Hopfield of the packet to the current SegID β'_i of the packet to get the SegID β'_{i-1}. Then the border router recomputes the MAC σ_i which depends on β'_{i-1} and checks whether σ_i and σ'_i are equal. If they are unequal, the path is invalid, and the packet gets dropped.

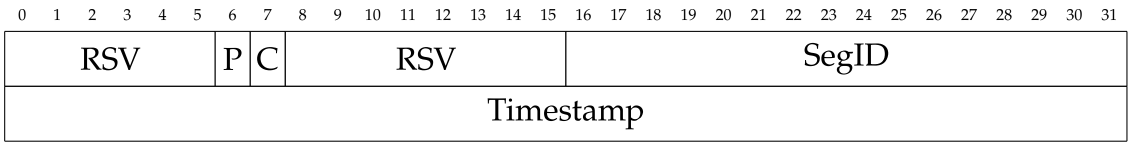


Figure 2.5: Info Field [3]

(Hop Field) The Hopfield (Fig. 2.6) contains information about the ingress and egress interface of the AS, which is the central part containing the information of the actual path. Whenever a packet arrives at a border router, it can forward the packet according to the ingress and egress interfaces of the Hopfield. Furthermore, it has a MAC, which, combined with the SegID, allows the border router to validate paths.

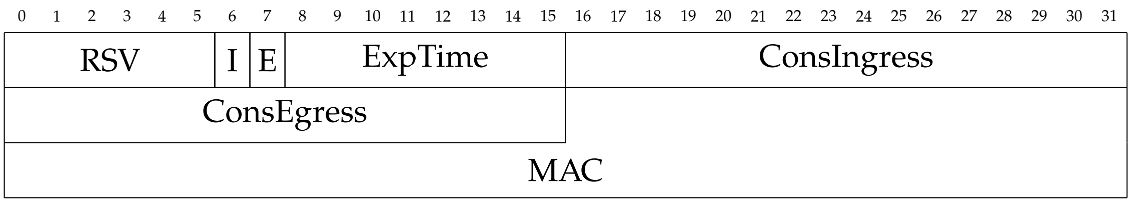


Figure 2.6: Hop Field [3]

2.2 Temporal Lensing

The idea of Temporal Lensing is copied from a technique used in the military called Time On Target (TOT). TOT describes the coordination of many artillery weapons to fire their rounds so that they hit the target at approximately the same time. The military standard of such a coordinated attack is to have the rounds hit the target around plus or minus three seconds from the prescribed time of impact. The reasoning behind this approach is that it had been found during WW1 that most casualties from artillery bombardment occurred during the first couple of seconds, where the enemy soldiers were still in the open. After the first impact, when soldiers found cover, the casualty rate drastically fell, making artillery bombardment over a long time a lot less effective.

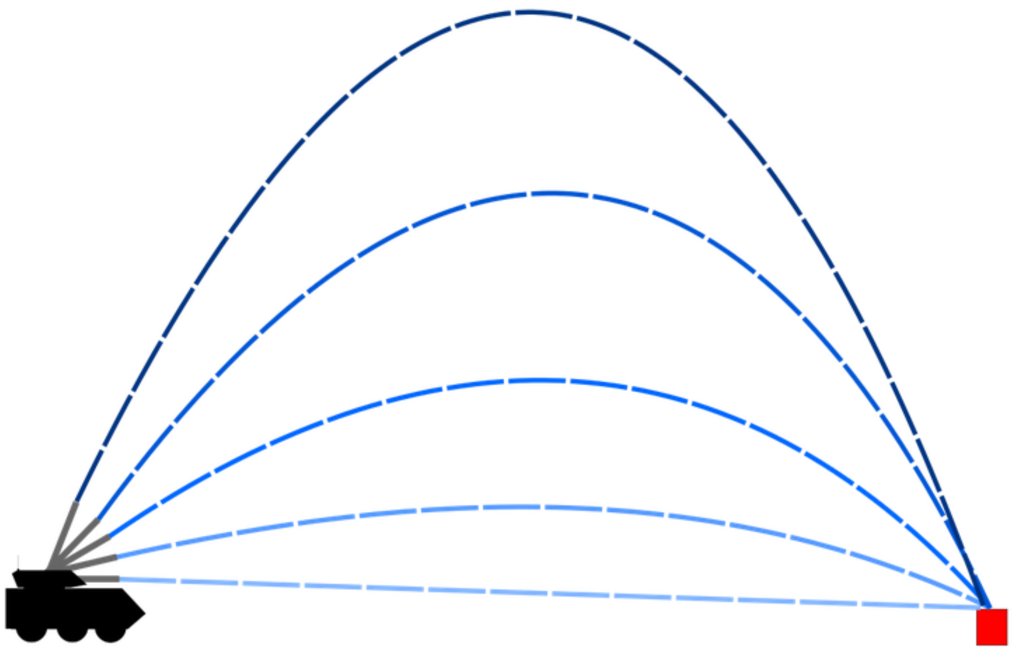
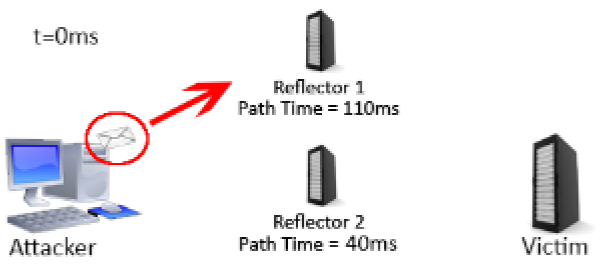


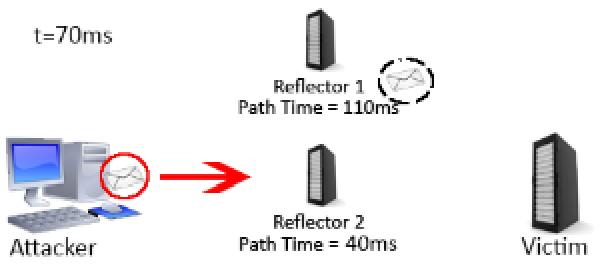
Figure 2.7: from https://github.com/PapaJoesSoup/BDArmory/issues/635

In a more sophisticated variant (Fig. 2.7), called *Multiple Rounds Simultaneous Impact* (MRSI), one artillery unit varies the angle of fire and velocity to have different flight times to the target. Thus enabling one single Unit to have a similar effect to multiple Units using (TOT).

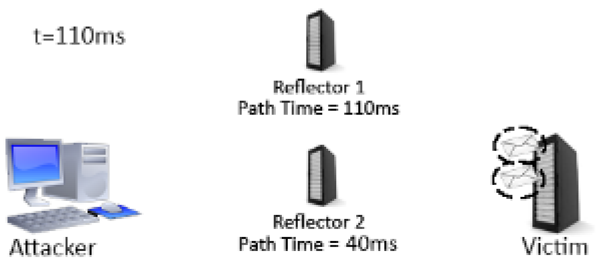
By borrowing the MRSI method, one can create a more refined version of a DOS attack. The idea is to send packets on different paths that vary in latency. The attacker wants to schedule its sending so that it starts by sending packets on paths with higher latency and then progressively switches to paths with lower and lower latency. Using this method causes the packets to arrive in a small time window, creating a sharp spike in bandwidth at the target. This spike can fill the



(a) At $t = 0\ ms$, the attacker sends one packet towards reflector 1



(b) At $t = 70\ ms$, the first packet is about 60% along its path to the victim and the attacker sends another packet to reflector 2



(c) At $t = 110\ ms$, both packets arrive at the victim

Figure 2.8: [9]

buffer and cause packet loss due to buffer overflow. In today's Internet, which uses destination-based routing, it is not directly possible to chose different paths when sending a packet to the victim. A workaround is to use a so-called reflection attack, where the attacker spoofs the destination address and sends a query to a DNS server. The DNS server then replies to the spoofed address which belongs to the victim. By querying DNS servers with various latencies to the victim, it is possible to coordinate the packets to arrive at the same time as seen in Figure 2.8.

The primary motivation behind the temporal lensing attack is to create relatively much disruption at a target server compared to the bandwidth available to the attacker. More formally, the primary metric to measure the efficacy of a temporal lensing attack is called *bandwidth gain*, which is defined as the bandwidth spike at the receiving server divided by the bandwidth available to the attacker.

2.3 Spoofing

Spoofing is the act by which an entity successfully identifies itself as another by falsifying data, which is still prevalent today [7]. Formerly spoofing was mainly used by attackers to falsify source addresses of IP packets to impersonate another host. Nowadays, it is used for all sorts of methods of authentication.

[11] There are different ways to detect falsified source addresses, called Source Address Validation Techniques. Source address validation (SAV) is commonly performed in network edge devices, such as border routers.

[11] One way to enact SAV is with access control lists (ACLs). Every border router keeps two ACLs: one for the ingress interface and one for the egress interface. These lists maintain a range of acceptable or unacceptable IP prefixes. Any packet with a source address that does not match the filter is dropped.

[11] Alternatively, one can employ Unicast Reverse Path Forwarding. The main idea behind Unicast Reverse Path Forwarding is that if a packet with destination address x is forwarded over interface i, a packet with source address x should arrive at the router over interface i. There are two main modes: strict and loose. In strict Unicast Reverse Path Forwarding, if a packet with source address x arrives over interface i, a packet with destination address x must leave the router over interface i; otherwise, the packet is dropped. In loose Unicast Reverse Path forwarding, if a packet with source address x arrives over interface i, the address x has to appear in the routing table, whereas the default route is not included.

One way in which IP spoofing can be used to inflict damage is by using a so-called reflection-amplification attack (Fig. 2.9). The attacker impersonates the victim by sending queries to a DNS server with the victim's source address. The DNS-server thinks that the queries came from the victim and sends all replies to the victim (Reflection). The Amplification part comes from the fact that DNS-server responses are usually way bigger than queries, especially if the attacker finds an Internet Domain that is registered with many DNS records. By querying the

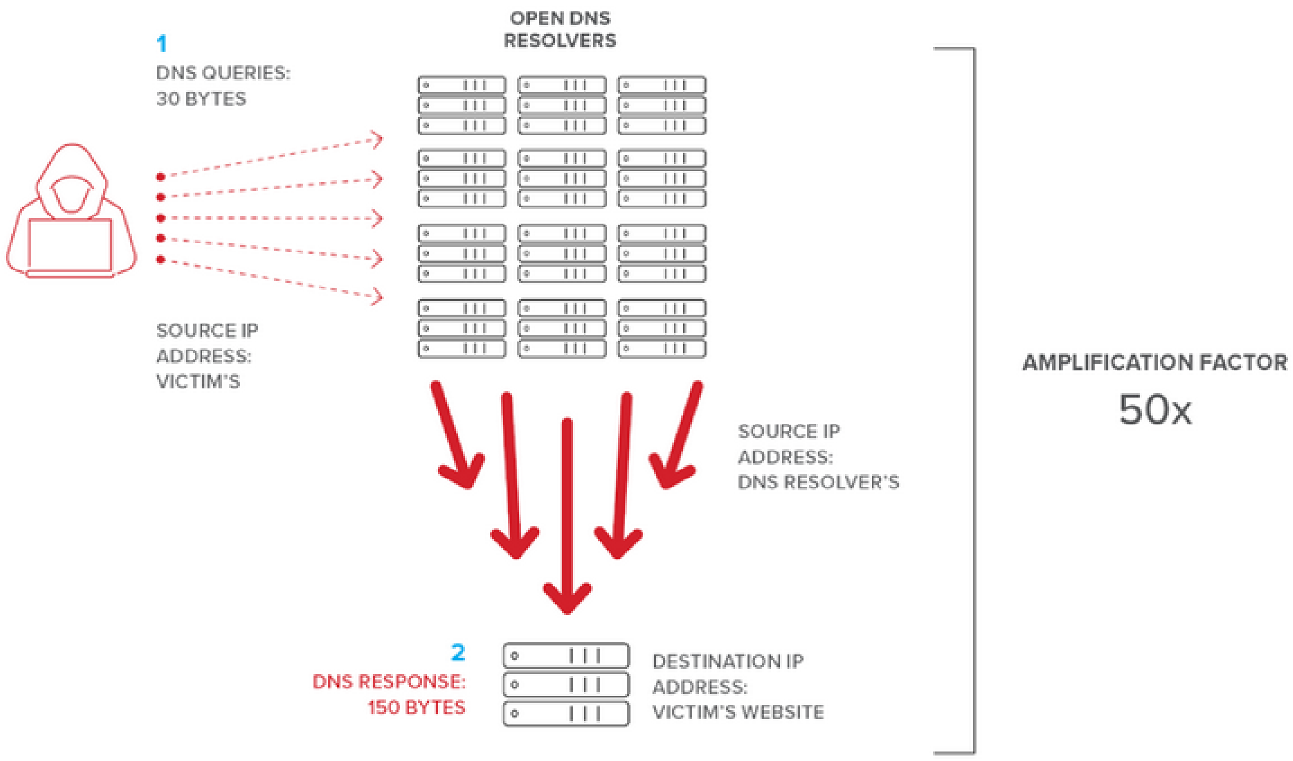

OPEN DNS RESOLVERS
1
DNS QUERIES: 30 BYTES
SOURCE IP ADDRESS: VICTIM'S
SOURCE IP ADDRESS: DNS RESOLVER'S
2
DNS RESPONSE: 150 BYTES
DESTINATION IP ADDRESS: VICTIM'S WEBSITE
AMPLIFICATION FACTOR
50x

Figure 2.9: https://cyberhoot.com/cybrary/dns-reflection-attack/

DNS-server server to return all records corresponding to the queried Domain, the attacker can significantly amplify the amount of Data the victim receives.

2.4 QUIC

[4] QUIC (Quick UDP Internet Connection) is a newer encrypted transport layer network protocol. QUIC was designed to make HTTP traffic more secure, efficient, and faster. Compared to TLS, which is implemented on top of TCP, QUIC is implemented on top of UDP.

[4] QUIC is a low-latency transportation protocol often used for apps and services that require speedy online service. This kind of protocol is a necessity for gamers, streamers, or anyone who relies on VoIP in their day-to-day life.

[4] QUIC brings the following changes compared to TLS:

1. Reduced connection times. To establish a TLS connection the client needs to perform a TCP handshake first, followed by a TLS handshake. QUIC only needs a single handshake (Fig. 2.10).
2. HTTP/2 on TCP can suffer from head-of-line blocking, a phenomenon where a line of data packets can be held up by the first packet. If one data packet is lost, the recipient must wait for it to be retrieved, which has a huge impact on connection performance. The QUIC protocol solves this problem by allowing streams of data to reach their destination independently. They no longer need to wait for the missing data packet to be repaired.
3. Stable connections when networks are changed. If you are connected to a web server via TCP and your network suddenly changes (from Wi-Fi to 4G, for example), each connection times out and needs to be reestablished. QUIC allows for a smoother transition by giving each connection to a web server a unique identifier. These can be reestablished by simply sending a packet rather than establishing a new connection, even if your IP changes.

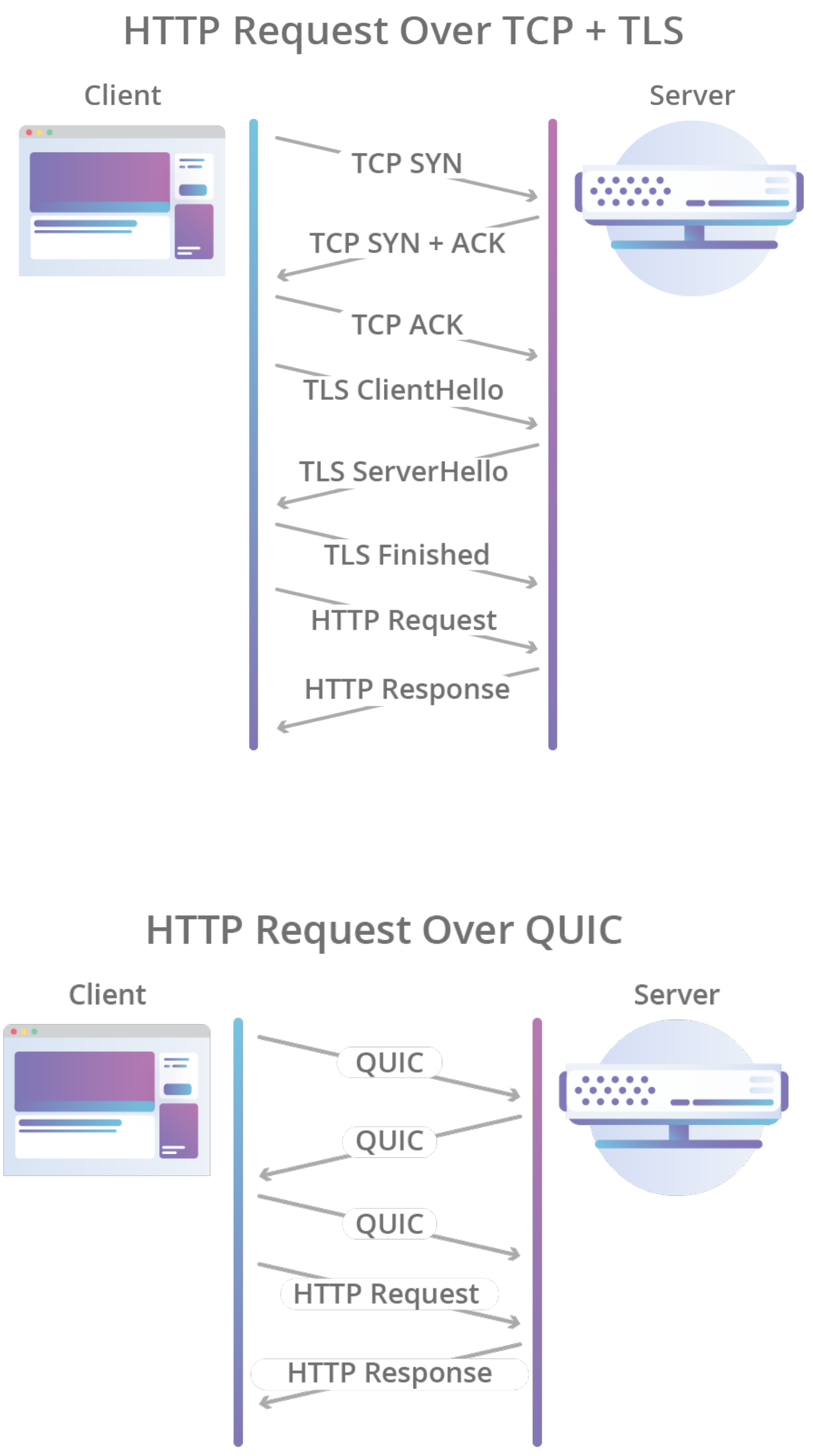


Figure 2.10: from https://blog.cloudflare.com/the-road-to-quic/

3 Problem Statement

This thesis aims to implement a temporal lensing attack and spoofing attack in the SCION network and measure their effectiveness.

For temporal lensing, one important metric is the bandwidth gain (Sec. 2.2). We are interested in finding out what factor of bandwidth gain is achievable in the current SCION network using a temporal lensing attack. Additionally, we will compare the distribution of arrival times of the packets at the victim using a temporal lensing attack compared to a simple DoS attack.

Concerning spoofing, the evaluation will be primarily a proof of concept.

3.1 Temporal Lensing Attack

3.1.1 Desired Properties

After specifying a target server, the attacker should retrieve information about different paths in the SCION network between attacker AS and victim AS and create a schedule of attack accordingly. An attack schedule defines at what time the attacker sends packets over which SCION path. While the attacker is sending packets on different paths, according to the schedule, to the victim server, the victim server should notice spikes in bandwidth. These spikes should be a multiple of the bandwidth available to the attacker.

3.1.2 Threat Model

We are assuming an attacker who only controls his host inside a SCION AS. He has standard SCION capabilities like querying for paths and pinging different locations. Additionally, we assume that there are multiple paths from the attacker's AS to the victim's AS, as we would not be able to conduct a TL-attack otherwise meaningfully.

3.2 Spoofing Attack

3.2.1 Desired Properties

The attacking host should create a UDP connection to a server with a spoofed source address by fooling a server into thinking that he is inside an AS that does not exist in reality. Additionally to fooling the server, the attacker should also be able to intercept returning traffic from the server, which sets this spoofing attack apart from conventional spoofing attacks. Using this capability, we want to complete a QUIC handshake using a spoofed source address with the server. By doing many QUIC handshakes in parallel, we want to be able to deplete the server's resources and remain stealthy, such that the attack cannot be mitigated easily on the network or application layer using source address / AS blocking.

3.2.2 Threat Model

We are assuming an attacker who, additionally to his host, also controls his local AS, meaning that he can access and manipulate traffic at the ingress border router interface to his AS.

4 System Design

4.1 Temporal Lensing Attack

Abstract description of the algorithm:

1. Get all paths from attacker AS to victim AS
2. For each path send a specified amount of pings and measure the mean and variance of the latency
3. Sort the paths according to the variance and pick the n paths with the lowest variance, where n can be freely chosen
4. Sort the n paths again, this time according to the mean latency
5. Create a schedule by dividing the attack time into discrete time units (slots). In each slot all packets are sent over the same path. In the first slot we send packets on the path with the highest mean latency. If the following inequality holds we switch to the next path : $|m_i - t| \geq |m_{i+1} - t|$. m_i is the mean latency of path i and t is a timer that is initialized with the value m_0 and is decreased by the slot size every time we calculate which path to take for the next slot.
6. Once the attack starts, the attacker just sends packets based on the schedule. After completing one cycle, the attacker simply repeats the schedule, to generate consecutive spikes.

The key idea is to exploit the fact that a sender in SCION can choose over which path he wants to send a packet to its destination. That is a tremendous advantage for an attacker compared to the destination-based routing in the current Internet. If we have the option of choosing the path of a packet as a sender, we do not have to rely on DNS-Reflections (Sec. 2.2) like in the conventional TL-attack.

The attacker should measure each path many times (Step 2) to get rid of the jitter and have a robust mean latency. Additionally, he should calculate the variance and drop paths with high variance (Step 3). The reason is that in a TL-attack, packets should arrive approximately at the same time. If a path has a high latency variance, it is hard to predict when a packet will arrive, and the chance is high that it will not land in the spike window.

The idea behind Steps 4 and 5 is to start by sending packets on the path with the longest latency to the target and then progressively switch to paths with increasingly shorter latencies to the target. According to the research paper from Ryan Rasti et al. on temporal lensing [9], a schedule is optimal if it maximizes the probability that each packet lands in the specified spike window. The schedule does that by choosing different paths to send packets over for each time slot. We assume that the latency of each path follows a normal distribution with the measured mean and variance, and the variances of all the paths are roughly the same. Under this assumption for each time slot, the probability of a packet landing in the spike window is maximized by choosing the path whose mean latency is closest to the time slot.

Example: Let us assume we finished Step 4 and have three paths with the following latencies:

1. $m_0 = 60ms$
2. $m_1 = 50ms$
3. $m_2 = 30ms$

We fix the slot_size to be $10ms$. After Step 5 the attacker would have created the following schedule:

$[0, 1, 2, 2, 2, 2]$

During Step 6 the attack according to the schedule above would proceed as follows:

1. During the first time slot $t = 60ms$ we are sending on path 0.
2. In the next time slot $t = 50ms$ we are sending on path 1.
3. For the next 4 time slots starting from $t = 40ms$ we are sending on path 2 till $t = 0ms$.

One could argue for a strawman approach without an attack schedule, where the attacker calculates the paths to send packets to the victim on the fly during the attack. The obvious drawback of this approach is that too much time would be spent calculating paths compared to when the attacker would be sending packets.

4.2 Path-Extension Spoofing Attack

Abstract description of the algorithm:

1. Query a path from the attacker AS to the victim AS
2. Prepend an additional Hopfield of an imaginary AS to the path. This Hopfield can have bogus ingress-, egress interfaces and a bogus MAC.

3. Increase the Hopfield pointer and SegLen0 of the path by 1.
4. Create a connection using this spoofed path. We must have control of the border router to intercept reply packets that are returned by the server to the spoofed address.
5. We can now start the QUIC handshake using the connection with the spoofed address, which sends packets with the spoofed address and receives packets intercepted by the border router.

The key idea of the Spoofing attack is explained according to the network topology of Figure 4.1. We assume that the attacker Controls AS E and has a host u inside E. The target of the attack is v inside AS D. The attacker gets a legitimate path from E to D, namely {E,A,B,C,D}. Then it prepends an imaginary AS to this path (X, Y or Z). The reason we prepend an imaginary AS is that IP addresses are not globally unique in SCION. For example, we can spoof the address *X:(RandomIP)* and we are sure that the reply to such a packet will be forwarded through E. Now when the attacker sends a packet to v using address *X:(RandomIP)*, v sees the path {X,E,A,B,C,D} with source address *X:(RandomIP)*. v will send its reply by reversing the path to {D,C,B,A,E,X}. X does not exist; that is why the attacker needs access to the border router in AS E to intercept packets destined for AS X and send them back to host u. Using this mechanism, we can repeatedly complete QUIC handshakes with v and thus exhaust v's resources. To be even more efficient, we can complete multiple handshakes in parallel to reduce downtime when packets are in transit.

(See Sec. 2.1.2 for a refresher on header layouts) We can have random values for the Hopfield in step 2 because no border router ever checks this Hopfield. By increasing the current Hopfield pointer and SegLen0 of the path by 1 it seems for subsequent border routers like the first spoofed Hopfield was already successfully

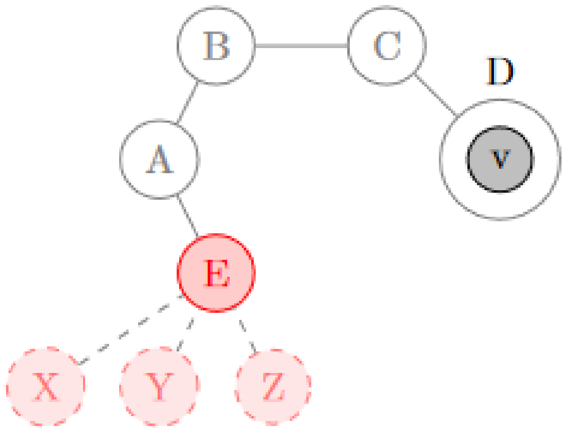


Figure 4.1

validated by another border router. We also keep the original SegID (Fig. 2.5) from the attacker AS to the victim AS; thus, all the subsequent border router checks will succeed.

5 Implementation

The following attacks were all implemented using Go.

5.1 Implementation TLA

Most functions we used during implementation were from the SCION base repository [6] by the network security group at ETHZ, or the appnet repository [5]. The appnet package provides a simplified and functionally extended wrapper interface to to the snet package of the former repository.

To implement the TL-Attack I created 3 separate files:

1. **ping_measurements** is responsible for getting latency measurements of each path from the attacker to the victim. This file implements step 1 and 2 of the abstract description of the TL-Attack described in Section 4.1
2. **tl_schedule** preprocesses the data measured in the previous steps and creates an attack schedule (step 3, 4, 5).
3. **tla1** starts the attack by sending packets on different paths (step 6) according to the generated schedule.

In the following the steps refer to the steps of the abstract algorithm described in Section 4.1.

5.1.1 ping_measurements

(Step 1) We start the measurements by calling the *TakeMeasurements* function, which begins by querying all paths from the attacker AS to the victim AS. To reduce the setup time, it is possible to specify how many paths the attacker should measure by altering the global variable *paths_to_measure*.

(Step 2) Next *TakeMeasurements* iterates over all paths and calculates the mean latency and the variance of the latency by sending multiple SCMP echo packets on each path. The two most crucial parameters are the number of measurements the attacker takes for each path *(attempts)* and the interval between each ping *(ping_interval)*. Taking more attempts increases the accuracy by decreasing the influence of jitter. By leaving the interval between two pings variable, it is possible to space them further out, which avoids congestion on the path to the destination.

It also increases the accuracy by taking measurements over a more extended period. If an error is encountered while measuring the latency of a path, the path is discarded. This information is then stored in a struct containing the path, the path's mean latency, and the latency variance.

5.1.2 tl_schedule

(Step 3, 4, 5) The attacker uses the path measurements as an input and creates a schedule that tells the attacker for each time slot over which path (connection) he should send the packets. The *TLschedule* struct has two attributes: *Slot_size* defines the lengths of one time slot of the schedule, i. e. , how long packets should be sent over one connection before switching to the next. *conn_slots* is an array that for each time slot contains a connection to send packets over during that time slot.

To create a *TLschedule* we call *createTLschedule*. First *createTLschedule* drops paths, where there has been an error during measurement (an example of an error might be loss of connectivity on a path because an interface is down).

Then the function *space_out* is called. *space_out* is called because the attacker wants to avoid a schedule where the differences between the latencies of the paths are too small. It is best to show the intuition behind spacing out paths on a small example:

Suppose we have a schedule where we only have three paths with latency $12ms$, $11ms$ and $10ms$. The attacker will send packets on $p0$ for $1ms$ then on $p1$ for $1ms$, and the rest of the time he will send packets on $p2$. Because the attacker only sends very few packets for the first two paths, likely, the packets from the different paths will not overlap at the specified time at the victim due to variances in the latency of the paths.

Space_out avoids schedules, where the path latencies are too close by looking for paths with similar latency and only keeping the paths with the lowest latency variance among them. By adjusting the global variable *min_distance*, it is possible to set the minimum latency difference of two paths in the schedule.

After spacing out the paths, the function *CreateTLschedule* sorts the paths according to their variance and picks the n paths with the lowest variance. (*keep_low_var* is a global variable that can be altered to keep the specified amount of paths). Next, the paths get sorted in decreasing order based on their mean latency, and then a path gets picked for each time slot according to the algorithm already presented in Section 4.1.

In an older version, the *TLschedule* struct was using a list of paths instead of a list of connections. During the attack, the attacker dials a connection using the appropriate path for each time slot. The drawback was a pretty significant overhead of around 1ms to dial a path, which compared to time slots of a couple of ms is quite a considerable fraction.

The duration of the time slot can be freely chosen. However, it should be noted that making the time slot too big can negatively impact the effectiveness of the attack because the way paths would be chosen would be too coarse-grained. There are almost no drawbacks to having smaller timeslots except more memory usage and longer preprocessing time. Since path latencies usually differ by around a couple of ms, going below time slots of 1ms does not benefit the attacker much more than time slots of a couple of ms.

5.1.3 tla1

(Step 6) To start the attack, we need to create a *TLA1* struct with attacker and victim addresses. The addresses are strings with the following format: `17-ffaa:1:eee,127.0.0.1`. 17 denotes the address of the ISD, `ffaa:1:eee` the AS address and `127.0.0.1` the host IP.

After creating the struct, we initialize the schedule using *InitSched*. This function uses the previous two files to create a schedule based on two parameters. *slot_size* describes how long we send packets on one path before switching to the next, and *ping_attempts* defines how many pings we send on each path to measure the average latency and the variance.

The actual attack is started by calling *StartAttack*. The core part of this function is the loop, which keeps track of the time and sends packets according to the current time and the attack schedule over the different connections. Connections that use the same path also use the same socket to send the packets. The current time slot is calculated by dividing the current time by the size of a time slot.

In an earlier version, we used *ticker* from the *time* package of golang, which caused the attacker to increase the current slot by one every time the ticker reached zero. The idea behind using a ticker is to avoid the overhead of calculating the current slot for each iteration, but it does not work at all. Using a ticker to transition from one slot to the next does not work because the ticker has an overhead of a couple of hundred microseconds, so it is not precise enough for this kind of task, which demands precision in the ms range. Furthermore, by constantly switching a couple of hundred microseconds too late, the delay accumulates, causing the attacker to use slots that are way behind schedule. Fortunately, all these problems were fixed by setting the current time slot as current time divided by the slot size.

5.2 Implementation Spoofing

The implementation of the spoofing attack is divided into three files:

1. **construct_spoofed** queries a path from the AS of the attacker to the AS of the victim and prepends an imaginary Hopfield to it (Step 1, 2, 3 of abstract Algorithm, see Section 4.2).

2. **nf_firewall** acts as our interface to the border router. It allows us to intercept the replies to the packets with a spoofed source address, which would not get forwarded to the attacker's host otherwise (Step 4).

3. **spoofed_conn** is an extension of the regular *snet.conn* class used in SCION, which allows us to send packets with spoofed source addresses and read packets by receiving the intercepted packets from the firewall (Step 4).

In the following the steps refer to the steps of the abstract algorithm described in Section 4.2.

5.2.1 construct_spoofed

This task is mostly about bit manipulation to create a semantically correct path. Paths are just byte arrays according to the specifications of Section 2.1.2.

(Step 1, 2, 3) After querying a path from the attacker AS to the victim AS, we pass the path to the function *add_spoofed_HF*. The function first looks at how many segments (Fig. 2.4) the path has. After learning about the number of segments the path has, we can create a new byte list with space for an additional Hopfield in front of all the Hopfields in the original path. (See Fig. 2.3 for the layout of a path). We copy the old PathMetaHdr and the InfoFields and then leave a 12-byte space for the spoofed Hopfield before adding the Hopfields of the original path.

We call the function *create_spoofed_HF* to create the actual bytes of the spoofed Hopfield. We could just set all the 12 bytes of the spoofed Hopfield randomly, without losing connectivity with the victim, because no border router will ever examine the spoofed Hopfield (see Sec. 4.2 for a more detailed explanation). The only reason the attacker might be motivated to add sane values is that the victim may examine the path. We set the ingress interface of the Hopfield randomly, which is the interface over which the PCB (Sec. 2.1.1) enters the AS during construction and set the egress interface to zero because the spoofed AS is a leaf AS.

At the end of the *add_spoofed_HF* function, after inserting the spoofed Hopfield, we have to make sure to increase the *SegLen0* and *CurrHF* by 1 (see Fig. 2.4 for an explanation of those two fields and Sec. 4.2 for why we have to increase those two fields by 1).

5.2.2 nf_firewall

In the implementation nfqueue [1] was used to intercept returning packets. To interface with nfqueue a repository implementing a golang interface to nfqueue was imported [2].

Nfqueue was originally intended to be used as a firewall, but it fit the spoofing task quite well, as it allows for easy access to all the packets that arrived at the border router. To do that, one needs to set up an iptable rule that puts all incoming packets into the nfqueue, which can then be inspected from the Go code.

(Step 4) The interception of the packets happens inside the *parse* function. Because in SCIONLab, SCION traffic is currently layered on top of UDP/IP, we first have to parse the IP and UDP packets before getting the SCION packet. Once we have parsed the SCION packet, the SCION firewall checks for the specific source address of the victim using the *intercept* function. If the source address of a packet matches the victim's address, the packet is intercepted, meaning that it is dropped from normal forwarding by setting the drop flag and instead put into a channel.

We create one channel for every QUIC connection we want to establish, i. e., all packets during the handshake of a QUIC session arrive at the same channel. After a QUIC session is established, there can still be keep-alive messages arriving on the channel, which then interfere with newer handshakes if one tries to reuse the same channels, that is why separate channels are needed.

The way packets get assigned to channels is based on the destination address, i.e., the address of the spoofed host. For example, if the address of the spoofed host is 1 the packet will be stored in channel 1. One might interject that the spoofing attack can easily be detected and averted, as only comparably small IP numbers are used. We should not forget that this is only a prototype, and the small numbers were used to simplify the program. This problem can be circumvented relatively easily in an actual spoofing attack by using a hash function. One idea might be to randomly choose the first 20 bits and use the remaining 12 bits to indicate the channel where the session should be established.

Another benefit of having many channels is that it is suitable for parallelization, which is crucial for an effective spoofing attack. With a strictly sequential spoofing attack, most of the time would be idly spent waiting for packets in transit to return during the handshake.

5.2.3 spoofed_conn

(Step 5) *Spoofed_conn* is an extension of scions *snet.conn*. It overwrites the read function by allowing the connection to read from the buffer, where the firewall stores the reply packets from the victim. Thus *Spoofed_conn* provides the same abstraction as a regular *snet.conn* while doing all the necessary spoofing work in the background. *Spoofed_conn* hides the complexity and allows users to implement spoofed connections for testing purposes transparently. This connection can then be used to complete a QUIC handshake.

6 Evaluation

6.1 Evaluation Setup

During the evaluation, we used two VMs, each hosting one SCION AS and a Host inside the AS. Each was assigned 2.5Gb of RAM and three processors from the host machine's Intel Core i7-4790K CPU. For the exact setup, refer to the Gitlab repository [12].

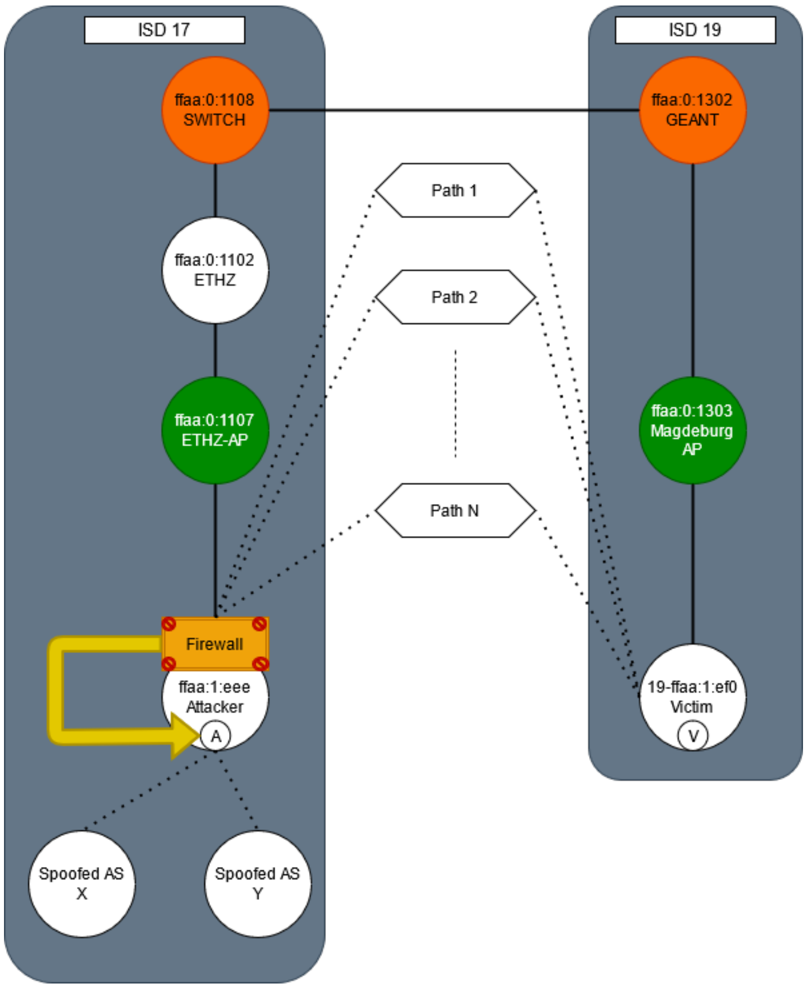


Figure 6.1

The attacker AS has the identifier **17-ffaa:1:eee** and is connected to the ETHZ-AP **17-ffaa:0:1107**. The victim AS has the identifier **19-ffaa:1:ef0** and is connected to the Magdeburg AP **19-ffaa:0:1303**. Figure 6.1 shows a strongly simplified version of the SCION topology, which only contains the ASes on the shortest path between the Attacker and Victim AS. Furthermore, the figure shows where the firewall during the spoofing attack is placed and how it forwards packets with spoofed addresses directly to the attacker host A.

6.1.1 Setup TLA

While doing the measurements of the TL-Attack, we had to artificially lower the attacker's bandwidth because many packets were lost due to buffer overflow. We lowered the bandwidth by pausing for $0.2ms$ after sending each packet. After lowering the bandwidth, we were able to send 40 packets per $10ms$. The packets were all identical and had a size of 25 bytes. We set the $slot\_size$ to 2 and the $min\_distance$ to 7 (see Section 5.1 for an explanation of those variables).

First, we wanted to compare the distribution of packet arrival times during a TL-Attack to a normal DOS-Attack where the attacker sends all the packets on the same path. We made this comparison for varying amounts of paths for the TL-Attack. To measure the results, we had the attacking host do one cycle (one spike) of the TL-Attack for a fixed amount of paths. The victim Host would then record the time in ms, from when it has started measuring, whenever a packet arrives, and return a CSV file with all the packet arrival times. We would record the number of packets sent and send the same amount of packets using a normal DOS-Attack and export this data as a CSV file. Using pyplot from python, we then plotted the arrival times of both CSV files. The diagrams use a bin size of $10ms$, meaning that each bar in the following diagram represents the number of packets, which arrived in this $10ms$ time slot. The diagrams are depicted in Figures 6.2 - 6.8, where the top diagram depicts the TL-Attack with varying amounts of paths and the bottom side depicts a normal DOS-attack. The average latencies of the paths used in the TL-Attack are listed below the diagrams.

One sees that in the TL-Attack case, packets arrive in a narrower time frame than the normal DOS-Attack. Furthermore, the TL-Attack has more significant and pronounced spikes, whereas the simple DOS-Attack diagrams have spikes of more or less constant height of around 60 packets per $10ms$ followed by gaps. These gaps also occur in the TL-Attack case and are likely caused by router interfaces, which buffer packets and forward them in bursts. However, the most crucial difference is that the maximum TL-Attack spike is higher than the regular DOS-Attack spikes.

(Fig. 6.9) Afterward, we also wanted to measure the bandwidth gain (See Sec. 2.2 for a refresher on bandwidth gain) and its change for varying numbers of paths. For each number of paths, we took five measurements and averaged them. It has to be noted that we were not trying to pin down the actual average bandwidth gain per number of paths used. Even when using the same number of paths, the efficacy of the attack still varies a lot. Reasons for this variation are how packets are forwarded in bursts at routers and the jitter on the paths themselves. This plot merely shows that the bandwidth gain is generally increasing with the number of paths used. It can be speculated that the increase is linear, which is feasible from a theoretical standpoint. Although to have certainty, one should look further into that, as there might be unknown constraints in reality. A linear increase in bandwidth gain would be very promising for the efficacy of TL-Attacks, especially in extensive networks, where there are hundreds of paths with significant latency differences.

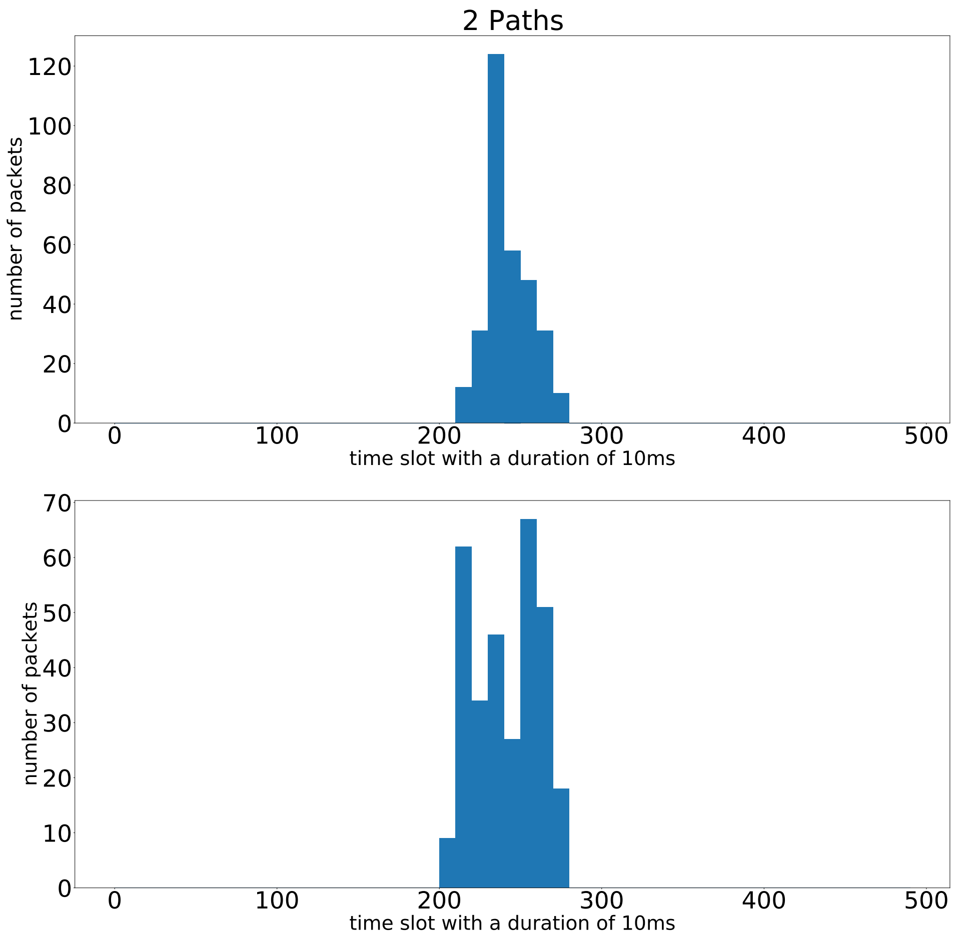


Figure 6.2: p0: 68ms, p1: 40ms

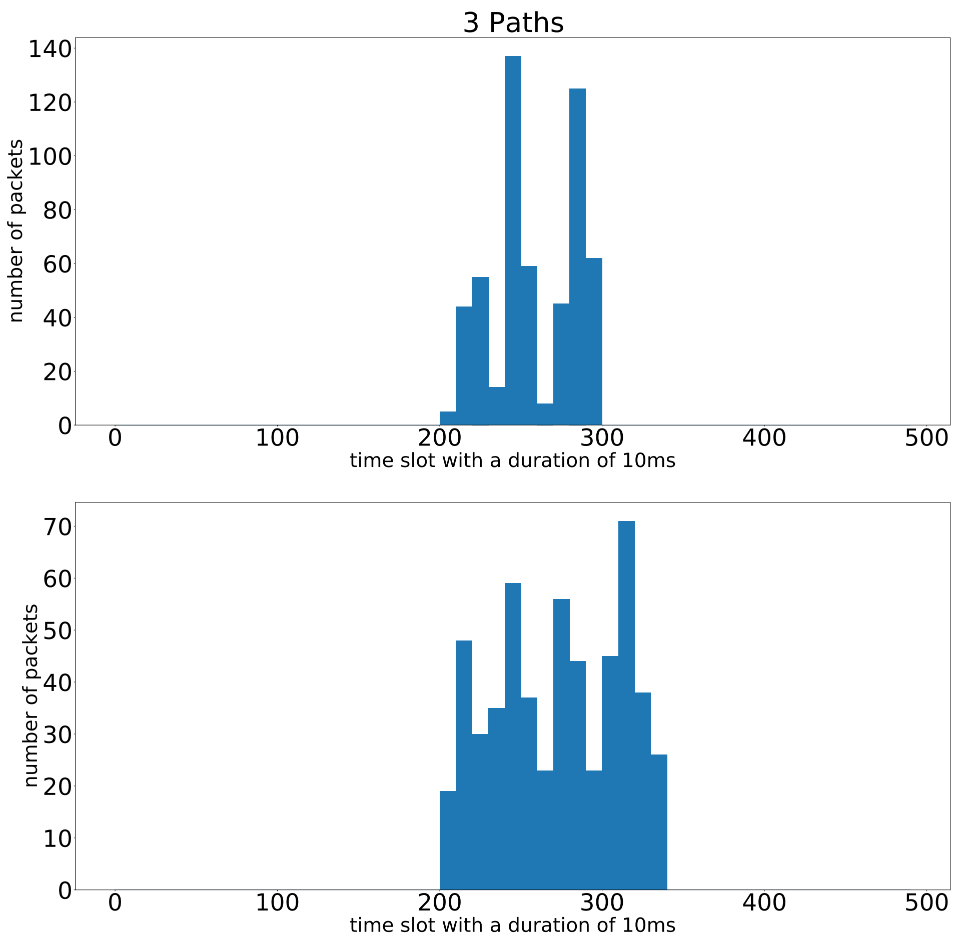


Figure 6.3: p0: 144ms, p1: 73ms, p2: 41ms

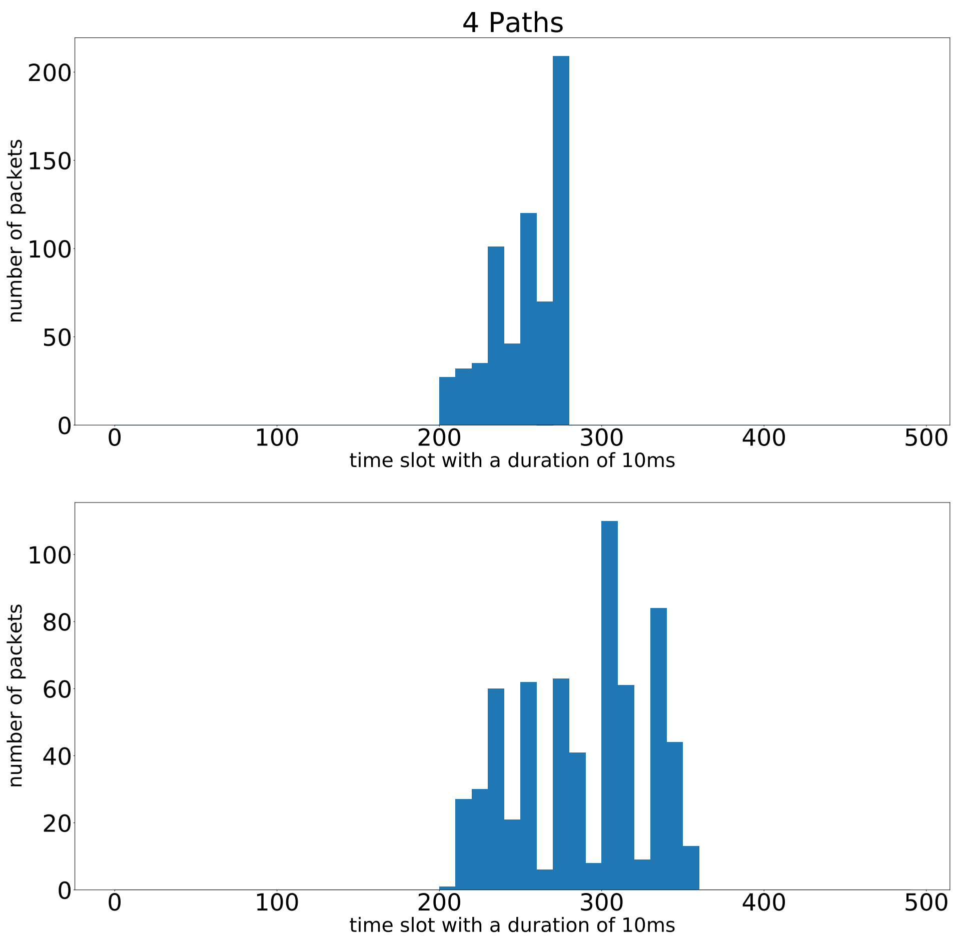


Figure 6.4: p0: 140ms, p1: 72ms, p2: 56ms, p3: 41ms

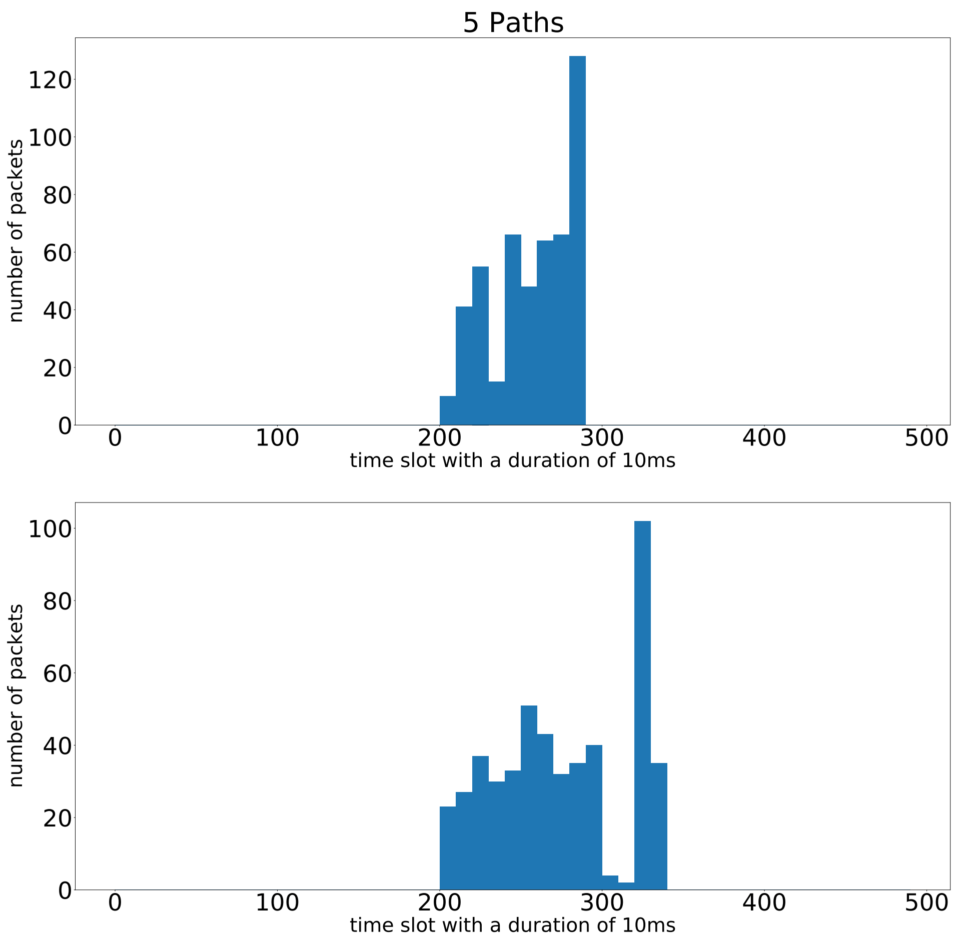


Figure 6.5: p0: 140ms, p1: 73ms, p2: 60ms, p3: 55ms, p4: 41ms

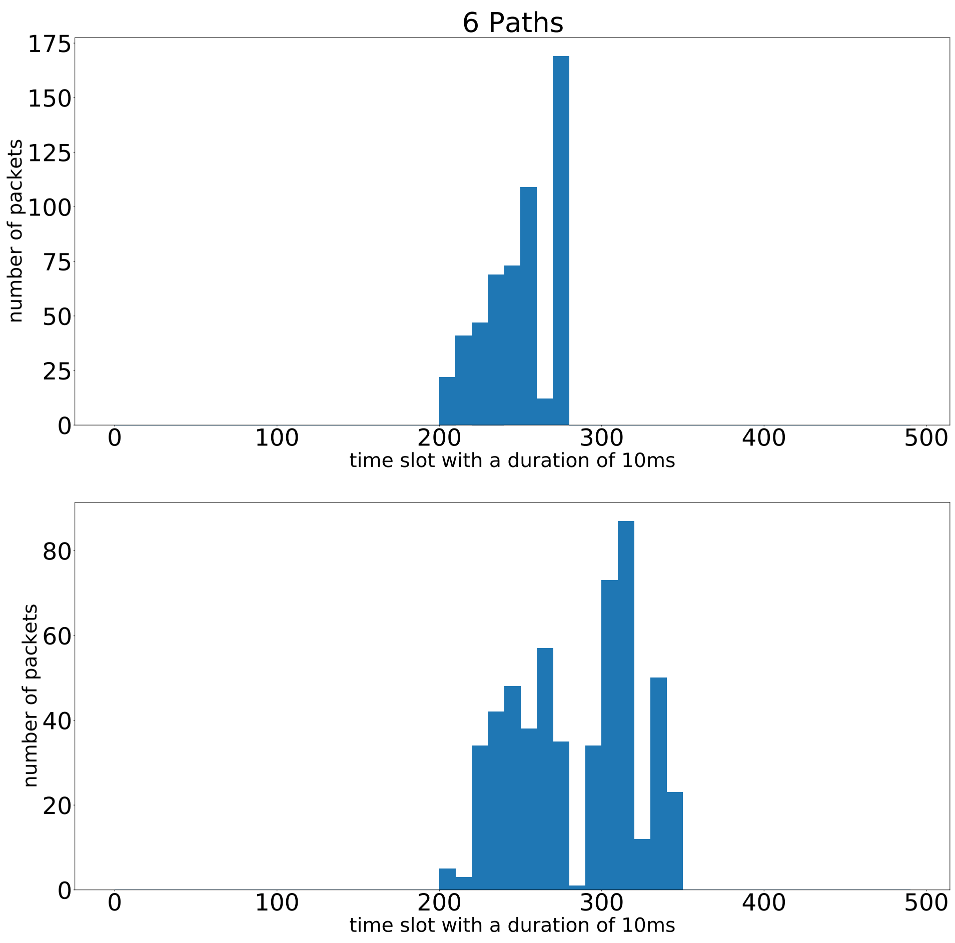


Figure 6.6: p0: 141ms, p1: 129ms, p2: 70ms, p3: 54ms, p4: 41ms, p5: 29ms

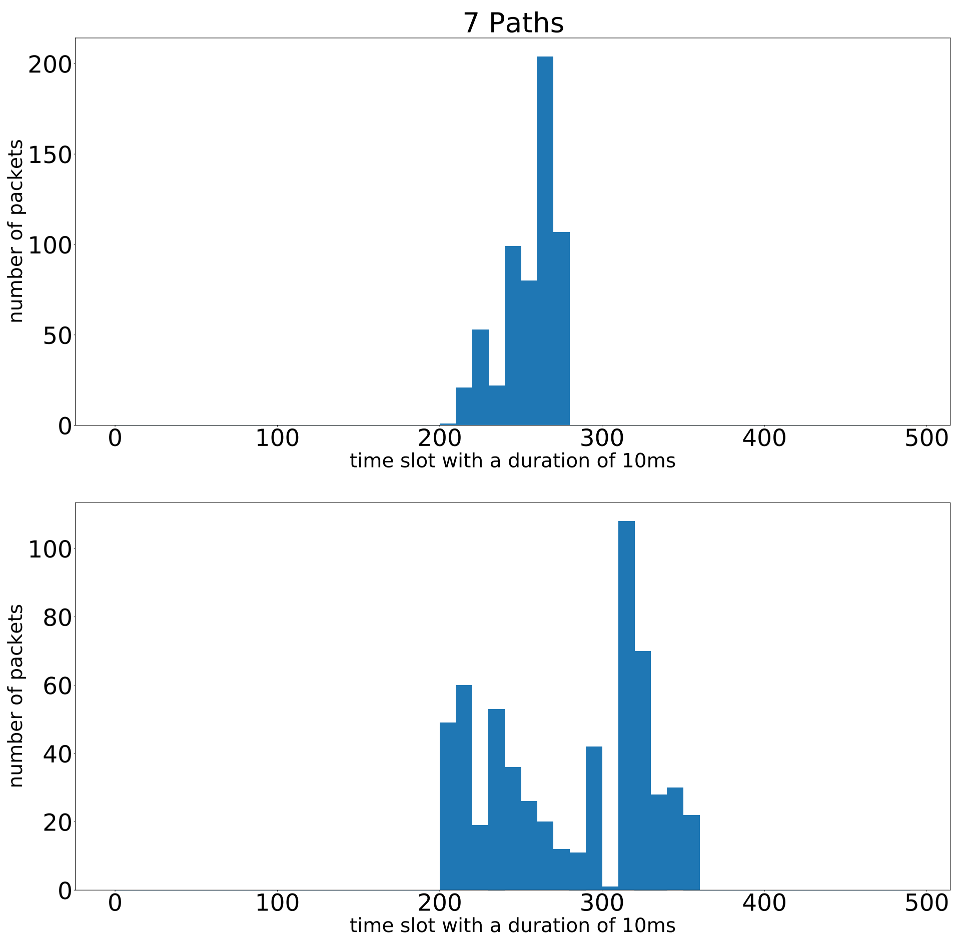


Figure 6.7: p0: 143ms, p1: 132ms, p2: 70ms, p3: 60ms, p4: 52ms, p5: 41ms, p6: 30ms

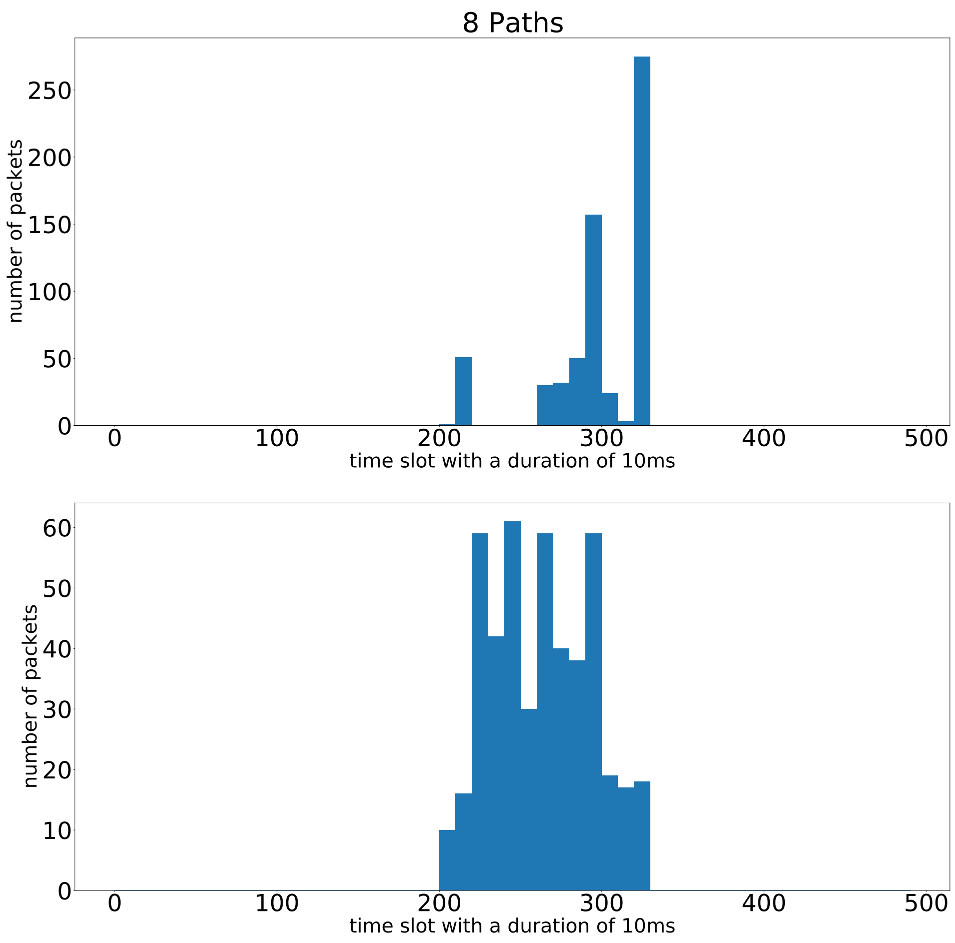


Figure 6.8: p0: 167ms, p1: 143ms, p2: 132ms, p3: 73ms, p4: 64ms, p5: 55ms, p6: 44ms, p7: 26ms

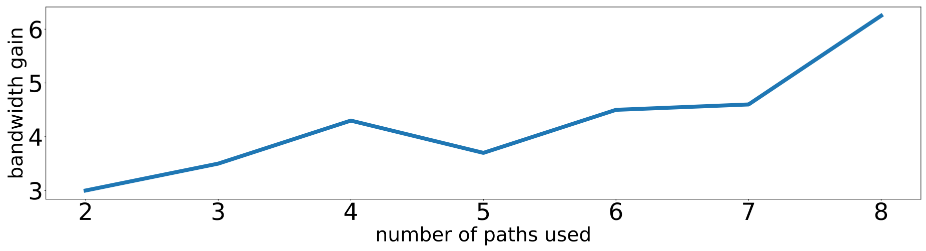


Figure 6.9

6.1.2 Setup Spoofing

To evaluate the efficacy of the spoofing attack, we measured how many QUIC sessions could be established per second. We also wanted to determine how much we could increase the speed at which QUIC sessions would be established by increasing parallelization.

We let each goroutine create 10 QUIC sessions and measured the total time it took for all goroutines to finish. For each amount of goroutines, we took five measurements and averaged them (Fig. 6.10).

One sees that the sessions established per second increase up to 80 goroutines, where we can create 180 QUIC sessions per second, after which it starts to drop slightly again. Overheads from managing multiple threads could impact the speed more than the time saved from waiting for a response during the QUIC handshake. It could also be possible that server-side issues are slowing down how fast new sessions are created when there are too many requests. This possibility is supported by the fact that the server would crash when we used more than 140 goroutines. It would certainly be interesting to look more into the effects on the server-side during such an attack, but we will leave that for future research, as we are primarily trying to prove a concept.

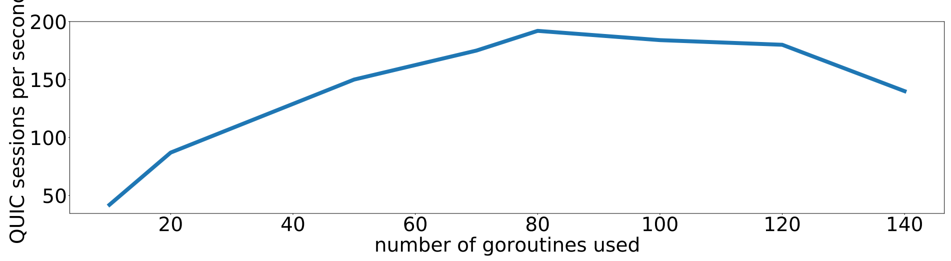


Figure 6.10

7 Discussion

7.1 TL Attack

7.1.1 Attack Impact

After evaluating the TL-Attack in Section 6.1.1, it seems that they succeed well in creating spikes at the victim server. Those spikes can be harmful to a server because there might be a buffer overflow during the spike on the server routers, causing legitimate packets to be dropped. The attack is especially disruptive if it creates many such spikes in a short time frame.

For that purpose, we propose another metric, in addition to *bandwidth gain* mentioned in the temporal lensing research paper published by Ryan Rasti et al. [9], to measure the efficacy of TL-Attacks called *attack period*. This metric measures the time needed to generate one such spike, or in different words, the duration of one TL-Attack cycle. Let us show based on an example why the *attack period* could be quite relevant. We assume that there are two different TL-Attack schedules $s1$ and $s2$. $s1$ has an *attack period* of $100ms$ and generates a spike with *bandwidth gain* 5, whereas $s2$ has an *attack period* of $500ms$ and generates a spike with *bandwidth gain* 10. If we only consider *bandwidth gain* $s2$ is the superior schedule, even though it could well be the case that a particular victim server would already experience buffer overflow with a *bandwidth gain* of 5. It would be much more disruptive for such a victim server to have several smaller buffer overflows than one huge one.

This also opens up ideas for algorithms that start with a schedule that creates a substantial *bandwidth gain* and adaptively scale down their *bandwidth gain* while decreasing their *attack period* as long as the victim server router still experiences buffer overflow. This could be checked by performing HTTP requests to the website, to estimate the responsiveness of the server.

7.1.2 Proposed Mitigation

It is central for a successful TLA attack to get accurate latencies for different paths. If one could somehow poison the RTT for the pings and create slightly off latencies, the TLA attack's efficacy would be greatly diminished.

(Idea 1) A first idea might be that the server alters the RTT by adding a constant, which is unknown, to the RTT of pings, so the attacker cannot learn the actual latency. Even though the attacker could not easily learn the actual latency, it would not diminish the efficacy of the TLA attack at all. By increasing or decreasing the RTT of all paths by a constant amount, the attacker would still be able to coordinate his attack so that all the packets arrive simultaneously. The only difference compared to knowing the actual latency would be that the time of the spike would be shifted by this hidden constant.

(Idea 2) Another idea might be to add jitter to the ping RTT. We might add jitter randomly drawn from a uniform or normal distribution and add it to the RTT, but this also does not prevent a TL-Attack; it merely slows the attack down. The attacker can circumvent this by sending many pings to the server, thus using the law of large numbers to get the latency without jitter. At least using random jitter, we increase the measurement phase of the attack.

Furthermore, it turns out that it is not possible to sample from some other arbitrary distribution to circumvent this problem. No matter what distribution we sample a jitter from, by the law of large numbers, we get as the average latency $l_i + \mu$. μ is the expected value of the distribution we sample a jitter from, and l_i is the actual latency of path i. $l_i + \mu$ is nothing different than what we have already seen in the first attempt, where we add a hidden constant to the RTT. The same argument holds for why sampling a jitter from an arbitrary distribution does not prevent a TL-Attack.

(Idea 3) A better solution is that every server uses a keyed hash function that takes as input the path the ping took and maps it onto a number $n \in [0, p]$ for some p. This n is then added to the RTT of the ping by buffering the ping request, so the attacker does not learn the actual latency. This method works better because the attacker cannot just send many pings to retrieve the actual latency. After all, the jitter always stays the same on each path. We also do not have the problem in case one, where the same hidden constant is added to every RTT. By having different jitters on different paths, it is hard for the attacker to schedule his attack so that all packets arrive in the prescribed window.

One way to possibly circumvent this is to create spoofed pings, where the attacker prepends Hopfields to the path (Sec. 4.2). Then the attacker sends many pings with spoofed paths to the AS of the victim. By the law of large numbers, we will again get an average latency $l_i + \mu$, where this time μ is the expected value of the hash function. Again the same argument holds as in idea 1, why the attacker can launch a successful TL-attack in this scenario.

A possibility for the AS of the victim to defend against such attacks is to apply source AS authentication, which will be discussed in Section 7.2.2.

(Idea 4) An interesting idea to consider is that each border router has a hidden constant (jitter_constant), which gets added to passing ping requests. The counter-argument for idea 1 does not hold here because different paths have a different jitter. This time every border router adds a hidden constant not just the server.

These ideas may sound good, but it is worth noting that adding latency to packets can be costly since the router needs to keep the packets in queues for that, which requires memory space. This is expensive on routers, which require high-speed memory that is not available in large amounts.

It is also worth mentioning that there is a reliable way for the attacker to get an accurate latency to a victim server, even if Idea 3 or 4 are employed. If the attacker has an allied host inside the victim's AS, he can use a custom protocol with the associated host to measure the latency to the victim, which the border router does not notice. That way, he can avoid all kinds of measures that try to obscure the actual latency by adding jitter to ping packets.

These methods, as mentioned earlier, mainly focus on disrupting the measurement phase by introducing jitter in a SCION-specific way. There are also other more general methods to defend against a TLA/DDoS attack which can be found in the research paper about temporal lensing from Ryan Rasti et al. [9].

7.2 Spoofing Attack

7.2.1 Attack Impact

The evaluation in Section 6.1.2 has shown that the spoofing attack is capable of rapidly establishing QUIC connections in parallel. The maximum amount of sessions per second we were able to establish with our setup (see Sec. 6.1 for the exact specifications) was around 200. Additionally, it might be noted that the speed can easily be doubled by doubling the PC's resources and doubling the number of goroutines, as there is no communication between the goroutines at all. It might seem that the *parse* function of the firewall might be the bottleneck (Sec. 5.2), as it sequentially checks incoming packets. However, the *parse* function can be easily parallelized by putting the whole function body inside a goroutine and immediately returning 0.

This attack is damaging to the victim server so far, as it rapidly drains the server's resources by reserving space for many bogus QUIC connections. The current algorithm only rapidly creates many QUIC connections without keeping them alive. In future work, one may want to compare slow loris attacks, which try to keep as many connections alive as possible, to the current attack. Maybe a hybrid model may also be possible, where the attacker starts by rapidly creating connections and then slowly transitions into a slow loris attack.

7.2.2 Proposed Mitigation

One way to avoid spoofing is to add a certificate and authentication (CertAuth) field to the path description. One possible place would be to add the CertAuth field between the *PathMetaHdr* and the *Infofields* (Fig. 2.3). This field holds a public key certificate and authentication of the source AS using the corresponding private key. By demanding such authentication, we would be unable to create imaginary ASes and spoof addresses in those ASes. Adding such a CertAuth field would not increase the payload of packets by much compared to the many Hopfields within the packets. The processing time would also not increase drastically because one only needs to check this (CertAuth) field once at the destination. It would also be beneficial to make it optional to check this field, so applications that are not at risk of being spoofed could forgo this check altogether.

A better method is to use the newly developed PISKES [10] system for source address authentication, which extends Ideas from *DRKey* (Dynamically Recreatable Keys). The primary motivation behind *DRKey* is to create an infrastructure, which allows to efficiently derive and distribute shared keys between routers and end hosts, which can then be used to authenticate source addresses. This is done by creating a hierarchy of keys, where keys from a lower level are derived from keys of a higher level.

Every AS generates a secret value SV, which is used to derive all subsequent keys. (Fig. 7.1) AS A generates SV_A and uses it to derive a symmetric key $K_{A->B}$ to communicate between AS A and AS B. AS A then passes this key to AS B. Using the symmetric key $K_{A->B}$ AS A and AS B can then derive symmetric keys that allow communication between hosts in AS A and B. All these keys are stored inside key servers inside the ASes. If a host wants to send a message to a host inside another AS, he has to first query the key server for the appropriate symmetric key.

The need to query key servers might open the door for another kind of attack. The attacker sends packets with random source addresses, for which the server then has to send many queries to the key server, trying to retrieve the appropriate key. It is possible to mitigate this attack by creating a partial key for a host H_A inside AS A. This partial key allows him to derive keys of the form $K_{A:H_A->B:X}$ for some Host X inside AS B, without querying the key server.

Using DRKey may work well to authenticate source addresses, but it also comes with a considerable performance cost for the AS. Thus it should not be used as a one-size-fits-all solution, and further research to explore different mitigation methods is still necessary.

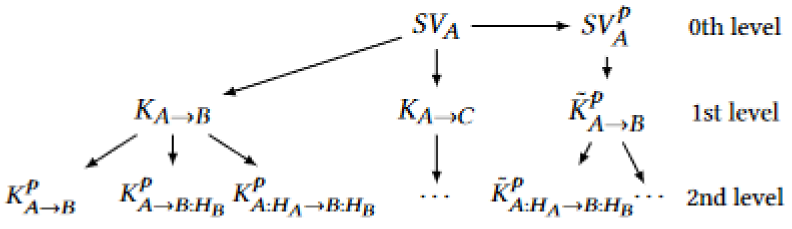


Figure 7.1: [10]

8 Conclusion

(Temporal lensing) This thesis has shown how to misuse the PCFS of SCION to create a temporal lensing attack. Furthermore, the evaluation in Section 6.1.1 indicates that the bandwidth gain scales linearly with the number of paths available in a network. This shows that the temporal lensing attack scales very well in extensive networks.

This thesis introduces a new metric to measure the efficacy of a temporal lensing attack (Sec. 7.1): *attack period*. In consideration of this metric, we propose a more refined version of a temporal lensing attack (Sec. 7.1).

In future work, one might want to extend this attack so that multiple attackers can coordinate their traffic to create one spike at a target. It would also be interesting to test this attack in a more extensive setting on an actual server, especially comparing the temporal lensing attack described in the design section to the more refined version in Section 7.1.

In conclusion, it can be said that temporal lensing attacks still pose a quite real threat, especially in multi-path networks like SCION. Currently there are still not many good solutions against temporal lensing (Sec. 7.1.2).

(Spoofing) We used path extension to create a spoofing attack, where the attacker can intercept the replies from the server. This capability allowed us to complete a QUIC handshake with a server. By parallelizing the handshakes, we created an attack, which aims to create QUIC sessions with a server as fast as possible, thus draining the server's resources. The evaluation in Section 6.1.2 showed that we were able to create up to 200 QUIC sessions with the specified setup (Sec. 6.1).

We then showed how one might mitigate such spoofing attacks using DRKey [10]. However, as already mentioned in Section 7.2.2 using DRKeys for authentication is by no means a one-size-fits-all solution, and there is still a need for further research for alternate methods of mitigation.

In future work, it would be interesting to compare our spoofing approach, which focuses on creating sessions as fast as possible, to a slow loris approach, which tries to keep sessions alive by sending packets on older sessions once in a while.

Bibliography

[1] nfqueu. https://home.regit.org/netfilter-en/using-nfqueue-and-libnetfilter_queue/.

[2] nfqueu golang. github.com/chifflier/nfqueue-go/nfqueue.

[3] Scion header. https://scion.docs.anapaya.net/en/latest/protocols/scion-header.html.

[4] E. Green. This is what you need to know about the new quic protocol. https://nordvpn.com/de/blog/what-is-quic-protocol/, 2020.

[5] N. S. Group. Scion apps repository. https://github.com/netsec-ethz/scion-apps.

[6] N. S. Group. Scion repository. https://github.com/scionproto/scion.

[7] F. Lichtblau, F. Streibelt, T. Krüger, P. Richter, and A. Feldmann. Detection, classification, and analysis of inter-domain traffic with spoofed source ip addresses. https://dl.acm.org/doi/10.1145/3131365.3131367, 2017.

[8] A. Perrig, P. Szalachowski, R. M. Reischuk, and L. Chuat. *SCION: A Secure Internet Architecture*. Springer, 2017.

[9] R. Rasti, M. Murthy, N. Weaver, and V. Paxson. Temporal lensing. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7163026&casa_token=V61s84wAFNQAAAAA:9sSNBgpRCSDAIyITUrhnWN_MRzOcfQesIWqDF2aAnn-5wVcg9Gc8cDt0XJZ8QtGs4sa3Ne7TDT8S.

[10] B. Rothenberger, D. Roos, M. Legner, and A. Perrig. Piskes: Pragmatic internet-scale key-establishment system. https://netsec.ethz.ch/publications/papers/piskes_final.pdf, 2020.

[11] K. Sriram and D. Montgomery. Resilient interdomain traffic exchange: Bgp security and ddos mitigation. https://en.wikipedia.org/wiki/Time_On_Target, 2019.

[12] J. Wanner and D. Lehmann. Bsc thesis dominik lehmann. https://gitlab.inf.ethz.ch/OU-PERRIG/theses/bsc_dominik_lehmann.

A An Appendix

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten Semester-, Bachelor- und Master-Arbeit oder anderen Abschlussarbeit (auch der jeweils elektronischen Version).

Die Dozentinnen und Dozenten können auch für andere bei ihnen verfasste schriftliche Arbeiten eine Eigenständigkeitserklärung verlangen.

Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuer und Betreuerinnen der Arbeit.

Titel der Arbeit (in Druckschrift):

Data Plane Security Aspects in Next-Generation Internet
Architecture Design

Verfasst von (in Druckschrift):
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Lehmann	Dominik

Ich bestätige mit meiner Unterschrift:
- Ich habe keine im Merkblatt „Zitier-Knigge“ beschriebene Form des Plagiats begangen.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu dokumentiert.
- Ich habe keine Daten manipuliert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Plagiate überprüft werden kann.

Ort, Datum	**Unterschrift(en)**
Münchringen, 15.9.21	D. Lehmann

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.