Diogo Carvalho Rodrigues

Bachelor Degree in Computer Science

Blocking DDoS attacks at the network level

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Engineering

Adviser: José Legatheaux Martins, Professor Catedrático,
Faculdade de Ciências e Tecnologia, Universidade Nova de
Lisboa

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

June, 2018

Blocking DDoS attacks at the network level

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Acknowledgements

I would first like to thank my thesis advisor Prof. José Legatheaux for the useful comments, remarks and engagement through the learning process for this master thesis. His door was always open whenever i had a question about my research or writing.

Also, i would like to thank my family for the continued support and encouragement throughout the process of learning and writing this thesis and my friends Duarte, João, Fouto, Joana, Prates, Mestre and Freire for witnessing the ups and downs of the whole process as well as providing vital feedback. Lastly, i thank my girlfriend Catarina who supported me throughout both emotionally and by acting as a secondary reader of this thesis.

Abstract

Denial of service (DDoS) is a persistent and continuously growing problem. These attacks are based on methods that flood the victim with messages that it did not request, effectively exhausting its computational or bandwidth resources. The variety of attack approaches is overwhelming and the current defense mechanisms are not completely effective. In today's internet, a multitude of DDoS attacks occur everyday, some even degrading the availability of critical or governmental services.

In this dissertation, we propose a new network level DDoS mitigation protocol that iterates on previous attempts and uses proven mechanisms such as cryptographic challenges and packet-tagging.

Our analysis of the previous attempts to solve this problem led to a ground-up design of the protocol with adaptability in mind, trying to minimize deployment and adoption barriers.

With this work we concluded that with software changes only on the communication endpoints, it is possible to mitigate the most used DDoS attacks with results up to 25 times more favourable than standard resource rate limiting (RRL) methods.

Keywords: Denial of Service, Spoofing, DDoS defense mechanisms, Cryptographic challenges

Resumo

Na Internet atual, os ataques de negação de serviço são um problema persistente e em constante crescimento. Estes ataques baseiam-se em métodos que inundam as suas vitimas com pacotes que estas não pediram, de modo a esgotar os seus recursos computacionais ou de largura de banda.

A elevada variedade dos métodos de ataque impõe dificuldades aos atuais mecanismos de defesa que não são completamente suficientes para os conter eficazmente. Todos os dias ocorrem uma multitude de ataques de negação de serviço, alguns até mesmo afetam a disponibilidade de serviços críticos ou governamentais.

Nesta dissertação, é proposto um novo protocolo de mitigação de ataques de negação de serviço que itera sobre tentativas anteriores e utiliza mecanismos comprovados, como técnicas de desafios criptográficos e marcação de pacotes.

A nossa análise das anteriores tentativas de resolver esta problemática, levou à concepção do protocolo desde a sua incepção, que se baseia em facilitar a sua adoção, necessitando o menor número de alterações possiveis à actual arquitetura da Internet.

Com este trabalho, conseguimos concluir que apenas com mudanças de software nos extremos em comunicação, é possível construir um protocolo que consegue mitigar os vetores de ataque DDoS mais usados, produzindo situações até 25 vezes mais favoráveis que as soluções atuais mais comuns baseadas em Resource Rate Limiting (RRL).

Palavras-chave: Ataques de negação de serviço, Falsificação de endereços de origem, Mecanismos de defesa a ataques de negação de serviço, desafios criptográficos

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The creation of the Internet was a turning point for the evolution of how we communicate with each other. In our day to day life, Internet access is everywhere and with the emergence of the Internet of Things[1], the future seems promising with regards to the continuous digital integration into our lives. However, certain design choices in the early days of the Internet have left it's mark in today's Internet security concerns.

One of these security concerns is related to the Denial of Service (DoS) attacks. This type of attacks goal is to degrade the availability of an Internet Service, effectively denying its normal functionality and impeding the access of legitimate clients.

1.1 Context

In a DoS/DDoS attack, we consider, the entity that intentionally blocks the normal Internet usage of a legitimate client to be the attacker in our system security model. As we will see in next chapters, we consider three types of attacks: reflection, protocol exploit based and brute force. The first makes use of public service provider machines to be used as reflectors in the attack, creating an amplification effect. For the second type, protocol exploit based attacks focus on depleting either computational, memory or bandwidth resources from the victim by exploiting flaws in a chosen protocol. For the last type, the attacker uses simple mechanisms and the attacks impact is caused by simple brute force.

Nowadays, a DDoS attack is at reach to most and not only to those with sufficient technical knowledge. According to [34], *Booters*, DDoS-as-a-service platforms, provide the illegal renting of time based DDoS attacks against a victim the buyer chooses, for only tens of dollars, for the most basic attacks.

[1] Simple, mass produced, Internet connected devices.

The Internet was designed under a stateless, best-effort model, ideal to build a simple and scalable communication substrate. For a message to be routed, the data is decomposed into packets by the sender, a source and destination address is attached and then those packets are injected into the network. Upon arrival at the destination, the packets are reorganized by the receiver into the original message. However, since each packet is treated individually by the network, its delivery is not guaranteed, e.g. if any problem occurs, a packet can be discarded.

The stateless, best-effort model has been at the heart of the explosive Internet growth, since it poses low barriers for the entry of new operators, new applications, new communication infrastructures, etc.

However, it has been implemented without any real concerns to what relates the certification of the authenticity of the senders address, also the willingness of the receiver to receive all packets directed to him is not required, as well as any innate message sending accounting mechanisms.

1.2 Motivation

Currently, the Internet has a very significant and real vulnerability against these types of attacks which need to be addressed. Any organization or individual with an online service can be anonymously disturbed by an entity with comparatively fewer resources. In an increasingly interconnected world, even governmental or critical services can be affected, or even compromised at the hands of cyber-terrorism.

Between February and March of 2015, a group of researchers setup 21 honey pots[2] and verified that 1.5 million DDoS attacks occured during these two months. All tried to use the 21 honeypot servers as reflectors for amplified attacks[14]. Reflection attacks will be further discussed in section 2.2.1. The problem is real and the attacks exist. As we will show later, the defence mechanisms are expensive and routinely fail to stop the attack.

1.3 Contributions

In this document we present the results of the research conducted on the topics of DDoS attacks and its current mitigation techniques.

We discussed the flaws that attackers exploit in order to achieve DDoS attacks capable of affecting the availability of sites on the Internet. On the other end of the spectrum, we also discuss the mechanisms that are currently deployed to try and mitigate these attacks.

In this work, we present a new network level DDoS mitigation protocol with a detailed description of the used mechanisms as well as a quantitative security analysis. The main goals of the protocol are to mitigate the most common known vectors of DDoS attacks, the protocol is based on the Host Identity Protocol (HIP), but it differentiates in the way

[2] A security mechanism set to appear as legitimate part of an attackers interested site, but is actually isolated and monitored. Similar to the police baiting a criminal.

that it aims to make the protocol's adoption easier while retaining the defining DDoS mitigation features of HIP.

To complement the new proposal and to prepare for future work, we also discuss the proposal's implementation and provide a proof of concept version.

1.4 Dissertation outline

The remainder of this work is organized as follows:

Chapter 2 We present the related work necessary for the better understand of the latter chapters of this dissertation. In particular, we delve into several major DDoS attack vectors by analysing and comparing them. Furthermore, we explain the current trends in DDoS defence mechanisms and how they will impact the future development of this work.

Chapter 3 An analysis on the Host Identity protocol [23] is provided, focusing on the protocol's innate DDoS mitigation mechanism as well as its possible limitations and barriers of adoption.

Chapter 4 We first introduce the End to end Light Security Protocol (ELSP), a new network level DDoS mitigation protocol based on proven mechanisms taken from HIP, and built from the ground up to be of easy implementation and deployment.

Chapter 5 We perform a quantitative analysis of the effectiveness of ELSP in face of several DDoS attack vectors.

Chapter 6 A discussion on the possible implementation barriers ELSP may face to be deployed in today's Internet, as well as discussion on a proof of concept implementation.

Chapter 7 The conclusion of this work, as well as the presentation of the future work we assume it requires.

Chapter 2

State of the Art

In this chapter we discuss several necessary topics relevant to the understanding of the goals of this thesis work. In particular, the following topics will be introduced:

In section 2.1 the basic principles of a DDoS attack will be covered.

In section 2.2 we will analyze the most important attack vectors used by today's attackers.

In section 2.3 we discuss the impact the Internet of Things(IoT) has on the current DDoS attacks.

In section 2.4 we discuss the current DDoS defense mechanisms available and we analyze a subset of the more relevant ones.

In section 2.5 we analyze and compare the current trends in DDoS attack vectors.

And in section 2.6 we summarize this chapter.

2.1 DDoS general notions

A distributed denial of service, or simply DDoS attacks, occurs when several malign computer systems flood the bandwidth or exhaust the computational resources of a targeted system, leaving its availability degraded.

A good analogy to better understand what a DDoS attack accomplishes is to think of it as if a large group of people were hired to try to enter the same door of a shop or business at the same time, leaving potential legitimate clients unable to enter the shop, thus disrupting its normal services.

In order to launch an attack of this type, the attackers need a great number of computers scattered around the internet. In this work, these computers will be referred as zombies since this this is the traditional jargon used in DDoS related literature.

Generally, a zombie is a computer system that has been infected with malware[1] and can be remotely controlled by an entity that isn't its owner. Dependent on the privileges of the installed malware, we will consider two types of zombies:

- User level access zombies - With only user level access, the zombie can't send packets with spoofed IP addresses, the attacker is thus limited in terms of which attack vector it can use for a DDoS attack. In the simplest scenario the attacker can use this type of zombie to flood the victim, e.g. with UDP packets or by trying to open many HTTP connections, this is generally called a brute-force attack.

- Root level access zombies - When the malware is installed with root privileges or by using privilege exploits, the attacker may have root level access on the infected system. For this type of zombie the possibilities are endless, since it can now use spoofed source IP addresses a lot of new DDoS attack vectors are open.

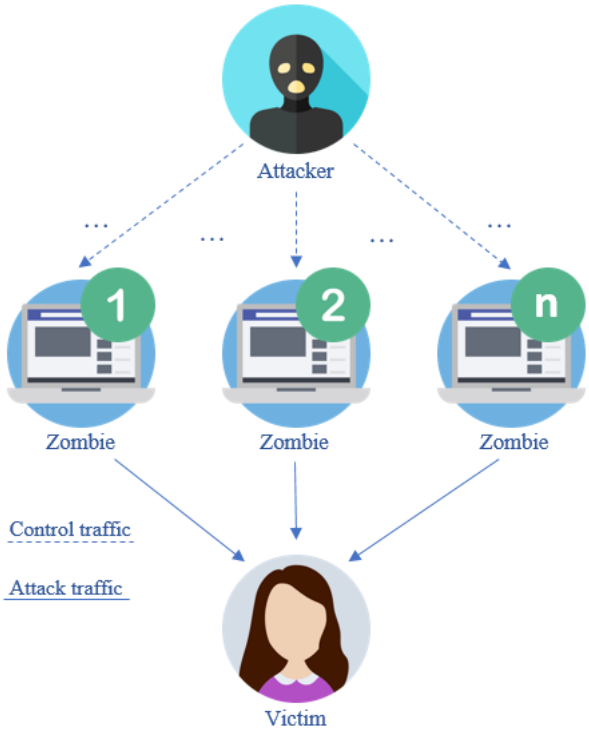

Figure 2.1: Architecture of a DDoS attack that uses a set of zombies to flood a victim.

[1] Malware, short for malicious software, is a term used to refer to a variety of forms of hostile or intrusive software, including computer viruses, worms, trojan horses, ransomware, spyware, adware, scareware, and other malicious programs.

2.2 Attack Vectors

In the next subsections, several types of DDoS attacks will be covered. DDoS attacks can be separated in groups in many different ways. In this work, in order to avoid complex taxonomies[20], DDoS attacks will be separated according to their most defining trait. This can either be the type of resource it mainly targets or the way its deployed.

As the years have passed, new ways to inflict a DDoS attack have appeared and their complexity and difficulty to overcome has risen sharply. In what follows we will concentrate on the most important attack vectors for each group.

2.2.1 Reflection attacks

In general, this type of attacks are performed using application protocols of the kind request/reply made over the exchange of UDP packets. A service is requested using an UDP packet whose source address is spoofed and has been replaced by that of the victim. This way, the reply will be sent to the victim. If the reply has more bytes than the request query, it then has an amplification factor greater than one and this is a fundamental factor in the effectiveness of a reflection attack.

Table 2.1: Examples of possible amplification factors of the studied amplified attack vectors

Protocol	Possible Amplification factor
DNS	70
CharGen	360
NTP	206
SSDP	30
CLDAP	57

In order to gather reflector servers, the attacker will need to scan the internet. To better evaluate the difficulty in gathering a large amount of reflection machines, Table 2.1 presents the results of an essay[29] to determine how long it took to find 1000 and 100,000 reflectors per protocol. This essay has been performed in 2014 and it's results can vary today but it still is important to get a glimpse of how easy it is for the attackers to assemble a set of reflectors in preparation for a reflection DDoS attack.

Therefore, we assume that for attacks using these protocols, the attackers gathering the necessary reflectors is not a issue difficult to overcome, even today. Some of these reflectors actions are performed because they use a ill-defined protocol, under the point of view of maturity (SSDP,NGP and CharGen) or because it is a necessary feature(e.g DNS).

In the following, we present the major reflection based DDoS attacks.

Table 2.2: Time taken to identify 1000 (T_{1K}) and 100,000 (T_{100K}) reflectors per protocol according to[29].

Protocol	Reflectors	T_{100k}	T_{1K}
DNS	7.782.000	92.5s	0.9s
CharGen	89.000	n/a	80.6s
NTP	1.451.000	195.1s	2.0s
SSDP	3.704.000	193.5s	1.9s

DNS In a DNS based Reflection attack[6] the reflection is achieved by eliciting a response from a DNS server using a spoofed IP address. Normally, DNS UDP responses are limited to 512 bytes, but when using EDNS[2] this number jumps to 4000 bytes, so when querying a DNS server with an ANY command, it then returns all known information to its DNS zone in a single response. An attacker can this way, reflect large amounts of traffic to an unsuspecting victim. DNS reflection attacks are very common and hard to detect because of the nature of DNS servers. With the increasingly popularity of DNSSEC[7], the amplification factor can even be larger because of the signatures used.

With a simple ANY query with only around 64 bytes, it was enough to produce an answer with 3223 bytes, an amplification factor of around 50x. Both the ANY query and the response can be seen bellow:

Query:

```
dig ANY isc.org @x.x.x.x
```

Response:

```
; <<>> DiG 9.7.3 <<>> ANY isc.org @x.x.x.x
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id:5147
;; flags: qr rd ra; QUERY:1, ANSWER:27, AUTHORITY: 4, ADDITIONAL: 5

;; QUESTION SECTION:
;isc.org. IN ANY

;; ANSWER SECTION:
isc.org. 4084    IN    SQA ns-int.isc.org. hostmaster
```

[2]EDNS is a specification for expanding the size of several parameters of the Domain Name System (DNS) protocol

```
  ... ...    ...    ...
isc.org 4084    IN    NS sfba.sns-pb.isc.org.

;; AUTHORITY SECTION:
isc.org. 4084    IN    NS ns.isc.afilias-nst.info
isc.org. 4084    IN    NS ams.sns-pb.isc.org.
isc.org. 4084    IN    NS ord.sns-pb.isc.org.
isc.org. 4084    IN    NS sfba.sns-pb.isc.org.

;; ADDITIONAL SECTION:
mx.ams1.isc.org. 484    IN    A 199.6.1.65
mx.ams1.isc.org. 484    IN    AAAA 2001:500:60::65
mx.pao1.isc.org. 484    IN    A 149.20.64.53
mx.pao1.isc.org. 484    IN    AAAA 2001:4f8:0:2::2b
_sip._udp.isc.org. 4094    IN    SRV 0 1 5060 asterisk.isc.org.

;; Query time: 176 msec
;;SERVER: x.x.x.x#53(x.x.x.x)
;; WHEN: True Oct 30 01:14:32 2012
;; MSG SIZE  rcvd: 3223
```

NTP The NTP protocol is used to synchronize the clocks of computers. It has a command called `monlist` that is used to launch NTP based reflection DDoS attacks[31]. When sending this command to a NTP server with a spoofed IP address, the server will send the target a list with the last 600 users that have requested the time from this server, this generates a response 550 times larger than the request query.

CharGen Chargen is a legacy protocol intended for testing, debugging and measurement purposes. When a client sends a UDP datagram to a server that exposes the CharGen protocol, the server generates a response containing a random number of characters (between 0 and 512) and sends them to the client. Its easy to see that the protocol can be used to launch reflected attacks with very high amplification factors.

SSDP SSDP is a protocol used to advertise and discover network services. It's a part of the Universal Plug and Play, or simply UPnP, protocol standard and comes enabled in millions of devices like web cams, routers, TVs and printers. In the first step of this attack, a SOAP request (M-SEARCH) with a spoofed IP address is sent to a UPnP enabled device. By receiving a M-SEARCH request the device responds with a XML file containing its location, operating system, UUID, etc. The more information the device responds with the larger the amplification factor of the attack.

CLDAP Connection-less Directory Access Protocol[42] was developed to support applications which require access to small amounts of information remotely. The protocol avoids the overhead of initializing and closing a connection, and works entirely in a connectionless fashion. A CLDAP server is normally used internally and use some form of authentication, but despite best practices, they are exposed to the Internet and coupled with the fact that normally these enterprise-grade servers are backed by computationally strong hardware makes this protocol an easy target for a reflection DDoS attack. Even though this protocol was retired from the IETF[3], many enterprises still use it today.

There are other types of reflection attacks and more can be discovered at any time. Reflection attacks can be performed whenever a zombie is able to use the victim's address as a spoofed address.

We will now turn to other types of attacks.

2.2.2 Protocol exploit based attacks

These type of attacks use certain vulnerabilities found in protocols. More often than not, these vulnerabilities aren't easy to overcome and are part of the definition of the protocol.

UDP Fragmentation UDP Fragmentation is based on depleting the computing resources of its target. This attack is based on exploiting the process of IP fragmentation and understanding this attack means understanding this process. When the size of a datagram is larger than the Maximum Transmission Unit (MTU) of a given link, the datagram is broken into smaller pieces, this smaller fragments are reassembled by the receiving host, which requires computational resources and memory. An UDP Fragmentation attack is based on exploiting this necessary process in order to cause pointless computations on the target. This can be achieved by using larger than the MTU value sized UDP datagrams to force fragmentation, and since this datagrams contain fake crafted data, the target server will be consuming CPU resources and fill memory buffers in a fruitless endeavor to reassemble useless datagrams[4].

SYN Flood A SYN flood attack exploits the design of the three-way handshake TCP Communication process between a client and a server in order to deplete its targets computing resources. In this attack, SYN packets are sent to the target using spoofed source IP addresses. In a normal TCP handshake, the SYN packet is used to signal the server that the source of this packet wants to start a TCP connection, the server when receiving a SYN packet allocates resources pertaining to this connection. A SYN Flood's objective is then to exhaust the targets resources by starting connections

[3] Internet Engineering Task Force, developer and promoter of Internet standards.

[4] For this reason, automatic IP fragmentation by intermediate routers is now considered bad practice and is no longer supported in IPv6.

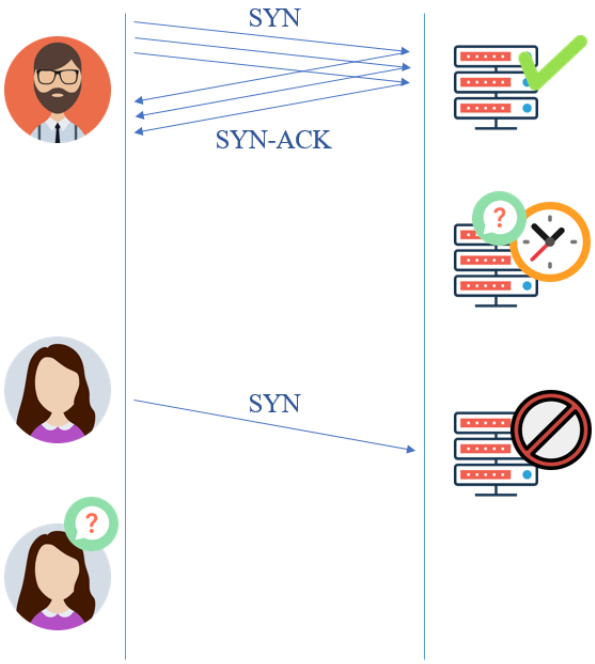

Figure 2.2: SYN Flood attack timeline: After the attacker has exhausted the server's resources, a legitimate request could not be fulfilled. Adapted from[38].

with random spoofed IP's or ports that will not be acknowledged thus leaving the connections half-open. A visualization of the effects of this attack can be found in Figure 2.4.

ACK Storm ACK Storm attacks[1] can be launched by a very weak Man in the Middle, or simply MitM, attackers, which can only eavesdrop occasionally and spoof packets. The Ack-storm attacks are based on the fact that, upon receiving a packet with the acknowledge number field (the receiver's sequence number) larger than the one sent by the receiving client, the client must, according to the TCP standard[26], resend the last sent acknowledgment packet to the other side, and discard the received packet. Using carefully crafted acknowledge packets for both sides of the communication, this results in the TCP connection between the client and server to be trapped in an infinite loop of sending and receiving empty acknowledgement packets. A basic example of an Ack-storm attack is as follows:

1. Pick up (at least) one packet from a TCP connection between a client and a server.
2. Generate two packets, each addressed to one party and with sender address of the other party (i.e. spoofed). The packets must be inside the TCP windows of both sides. The packets should have content - at least one byte of data.
3. Send the packets to the client and the server at the same time. The connection will then enter an infinite loop of sending ack packets back and forth between

both parties.

An example model of an ACK storm attack can be viewed in Figure 2.5.

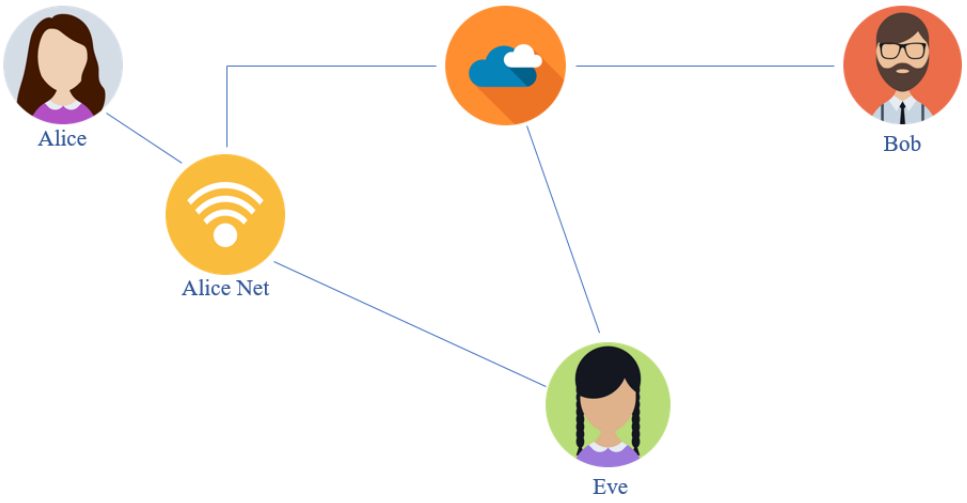

Figure 2.3: Example Ack-storm attacker model - Alice is connected through the wireless access point AliceNet, to a remote web server Bob. Eve is able to receive occasional traffic from Alice's network. In addition, Eve's ISP does not filter traffic, so Eve is able to send spoofed packets to the Internet. Adapted from [1].

Optimistic ACK An optimistic acknowledgment (opt-ack) is an acknowledgment sent by a misbehaving client for a data segment that it has not received. The Optimistic Ack DDoS attack[36] exploits the TCP congestion control system. In a normal scenario, TCP's congestion control adjusts the transmission rate according to the available bandwidth. The way for TCP to know how congested the communication channel is, is based on the Acknowledgement packets sent by the destination, if ACK's are being received, that means that the network isn't congested and the transmission rate can be adjusted to higher values. The way an Opt-Ack attack works is, the attacker sends many ACK packets without receiving the corresponding data. The server then rises the transmission rate to the attacker, congesting the servers uplink and saturating the whole path from the server to attacker. TCP implicitly assumes by design that remote clients generate correct ACK feedback and this incorrect feedback deteriorates end-to-end performance. However, an attack that doesn't care about data integrity can violate this assumption, forcing the server to send many packets into the network. In Figure 2.6 we can see a possible attackers algorithm to perform such an attack. The attacker need three parameters, if attacking more than one victim, a number n representing the number of victims to attack, the maximum segment size (mss) and the window scaling (wscale). The attacker then keeps track of each victims estimated window (W_i) and the sequence number to acknowledge (ack_i).

Attack($\{v_1 \dots v_n\}$, mss, $wscale$)
1: maxwindow$\leftarrow 65535 \times 2^{wscale}$
2: $n \leftarrow |\{v_1, \dots, v_n\}|$
3: **for** $i \leftarrow 1 \dots n$ **do**
4: connect(mss,$wscale$) to v_i , get isn$_i$
5: ack$_i \leftarrow$ isn$_i + 1$; $w_i \leftarrow$ mss
6: **end for**
7: **for** $i \leftarrow 1 \dots n$ **do**
8: send v_i data request { http get, ftp fetch, etc. . . }
9: **end for**
10: **while** true **do**
11: **for** $i \leftarrow 1 \dots n$ **do**
12: ack$_i \leftarrow$ ack$_i + w_i$
13: send ACK for ack$_i$ to v_i { entire window}
14: **if** $w_i <$ maxwindow **then**
15: $w_i \leftarrow w_i +$ mss
16: **end if**
17: **end for**
18: **end while**

Figure 2.4: Attacker's algorithm to perform an Optimistic Acknowledgment attack.From [36]

2.2.3 Brute force attacks

In this subsection we discuss the least complex forms of DDoS attack vectors. These methods mainly rely on simple protocols and don't require root level access zombies.

Direct UDP Flood A direct UDP flood is a very simple attack vector, it only needs user level access zombies and has no amplification factor. It simply consists of sending UDP packets to a target to try to exhaust its bandwidth.

HTTP Flood An HTTP Flood is a layer 7 DDoS attack in which an attacker sends HTTP requests to a target server. With this kind of attack, the HTTP clients such as web browser interacts with an application or server to send HTTP requests. The request can be either "GET" or "POST". The aim of the attack is then to compel the server to allocate as many resources as possible to serving the attack, thus denying legitimate users access to the server's resources. We then consider two different types of HTTP floods:

GET flood - The GET request is used to retrieve static content like images. Typically this induces relatively low load on the server per request.

POST flood - POST requests are more likely to require the server to perform some kind of processing, such as looking up items in a database. Therefore, HTTP POST flood attacks typically impose higher load on the server per request.

2.3 The impact of the Internet of Things

With the increasingly higher number of devices connected to the internet, the number of potential zombies to be used in an attackers botnet also rise. The term Internet of Things, or simply IoT, refers to the mass produced low cost devices with Internet connection. These devices profit margin is also very thin, and thus the security measures implemented in these devices are often left behind or of weak quality. Also due to the nature of these devices, applying security patches is not an easy task, which leaves the devices in an unsecure software version compared to the current known vulnerabilities.

A famous botnet known as Mirai, is estimated to be made of over 100 thousand zombies, of which mostly are hacked "Internet of Things" devices such as IP cameras, routers and internet connected sensors. In September 2016 a 620 Gbps attack [5] was launched by the Mirai botnet, one of the largest ever recorded DDoS attacks. As the Mirai malware source code has been made open source[6], many new variations have appeared, namely Persirai and many unnamed others. With the amount of IoT devices vulnerable, it's possible to create very large botnets which otherwise wouldn't have been possible to create without much more work from the attackers. This leads to a point where spoofing IP addresses is no longer needed and the DDoS attacks can be based on fully establishing TCP connections and, e.g , generating HTTPS floods.

2.4 DDoS Defense Mechanisms

DDoS attacks are inherently difficult to combat, their similarity to legitimate traffic makes defense and detection mechanisms hard to implement. DDoS attacks are not new and over the years there have been several proposals with different techniques, but for most of them to be implemented in production level, the internal foundations of the Internet would need to be modified and upgraded. For example, there are proposals that require complex software implementations on the core of the Internet, and for it to work every entity responsible for an important Autonomous Systems would need to collaborate and invest in this software changes and possibly also hardware changes. This leads to a problem called Tragedy of the commons[7], a good analogy to explain this situation would be: In the internet there is no centralized legislator and thus, creating laws in this environment is very hard to achieve. If there were no environmental laws that prohibit a certain level of pollution for a certain business, why would this business implement such measures that would only affect its profit? A single entity has no real value in implementing such measures if they are alone, if all of the polluting business implemented pollution

[5] https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos

[6] https://github.com/jgamblin/Mirai-Source-Code

[7] The tragedy of the commons is a phrase referring to a problem of shared resources. Increased use of the resource by any sharing member hurts all members equally. Yet the benefit to a member that uses more of the resource outweighs, to him, the damage from the overall increased use. As a result, all sharing members choose to maximize their use of the resource, resulting in its inevitable depletion

reduction measures everyone would at least gain from the fact that there would be no more pollution.

One of the main reasons DDoS attacks are so common is because the internet infrastructure industry allows IP spoofing on a large scale which enables attackers to be untraceable and target victims using legitimate reflector servers.

Another issue is related with the lack, by design, of security, accountability and authentication at the IP level. This is related with the "end-to-end arguments of system design"[33] which mandates a simple IP layer that cannot easily give many guarantees to the edge.

In these types of situations, botnets thrive because it's not the core of the Internet that checks whether or not computers send traffic from IP addresses that they have been assigned. Thus, in this work, mechanisms for DDoS defense, such as ones involving the change of the Internets core, will not be reviewed by lack of incentives to implement them.

2.4.1 Server replication

One simple way to protect a server from DDoS attacks, consists in replicating the existing infrastructure and making the attackers require more attack power to degrade the systems availability. This is not a rock solid solution, as it's easier for the attacker to get resources than it is for the defender. Many organizations lack the resources for this replication, because, as we have seen the attackers need less resources than the victims to successfully cause an impact.

However, a common technique consists in sharing the costs of that replication by using a Content Delivery Network(CDN)[40].

2.4.2 Absorption

An absorption technique called Secure Overlay Service (SOS)[12] prevents a DDoS attack in a proactive way. It's based on an architecture that is constructed using a combination of secure overlay tunnels, routing via consistent hashing and filtering. The defense is based on performing intensive filtering on network protected edges and pushing the traffic to the network core where high-speed routers can better handle incoming traffic, and by introducing a randomness and anonymity factor that makes it difficult for an attacker to target nodes in the path to a SOS protected destination.

To summarize, the sequence of operations in a SOS architecture is as follows:

1. A site installs a filter in its viscinity and then selects a number of SOS nodes to act as secret servlets, that is, a node that is allowed to forward traffic through the filter to that site. Routers at the perimeter of the site are only allowed to let traffic from these servlets to the networks internal core.

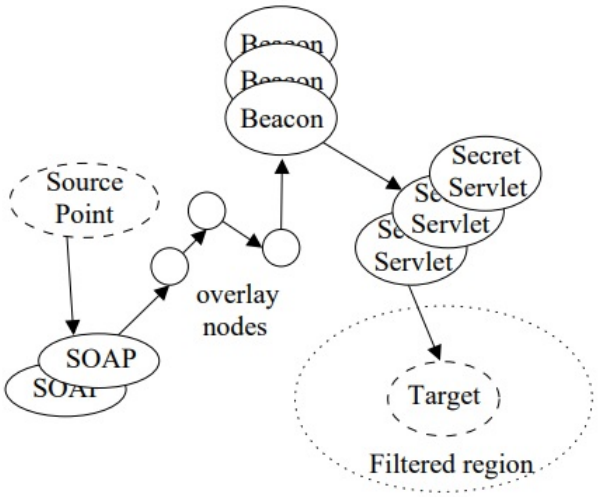

Figure 2.5: Basic SOS architecture. Adapted from [12].

2. When a SOS node is informed that it will act as a secret servlet for a site, it will compute a key k for each of a number of well known hashing function[8], based on the site's network address. Each of these keys will identify a number of overlay nodes that will act as a beacon for thaat site.

3. Then the servlets or the site will notify the beacons of its existence. A beacon, after verifying the validity of the request will store the necessary information to forward traffic for that site to the appropriate servlet.

4. A source that wants to communicate with the site needs to contact an Overlay Access Point(SOAP). After authentication, the SOAP routes all traffic from the source to the target site via one of the beacons.

5. The beacon routes the packet to a secret servlet through a filtering router to the target site.

In Figure 2.5 we can see a high-level overview of the SOS architecture that protects a target site to only receive legitimate transmissions.

Currently there are several providers for this type of service, making it a very effective way to protected high priority or emergency services with an overlaying network that can absorb the attacks. Examples of providers of this service are Akamai/Prolexic, Cloudflare and Incapsula.

2.4.3 DNS RRL

As seen in the earlier sections of this work, a DNS DDoS attack takes advantage of the fact that small DNS queries can generate a much larger response, and since the attacker spoofs the IP address of the victim to reflect the network traffic, it makes it difficult to

[8]Function that can be used to map data of arbitrary size to data of fixed size.

trace the attacker. Response rate limiting[30, 39] is a mechanism for limiting the amount of unique responses returned by a DNS server, thus limiting the effectiveness of a DNS reflection DDoS attack by dropping the packets that exceed the pre-configured rate limit. As an overview, DNS rate limiting works as follows:

- As the server generates a response to a DNS query, the requesters IP address is added into a bucket. For the most common implementation, there's one bucket per /24 subnet for IPv4.

- If the number of unique responses exceed the configured limit, the server starts dropping all requests for the specific subnet during a certain period.

With this overview it's easy to encounter a problem, if all requests are dropped for the whole network that may also affect legitimate users. To combat this flaw it's possible to configure a LEAK-RATE value of one or more. With a LEAK-RATE value of one, every dropped request generates a response with the TrunCation flag[21] set to one. This may still create a heavy load on the victim, as it's still a non-amplified DDoS attack. When the victim receives a response with control flag set to one, it then knows that they are under attack and in order to communicate with the DNS server it needs to establish a TCP connection.

Table 2.3: Several LEAK-RATE configurations for an ANY attack targeting a DNS server. From [39].

LEAK-RATE	False positives	In	Out	Amplification factor
1	0%	80KB/s	80B/s	1:1
2	50%	80KB/s	40KB/s	1:0.5
3	66%	80KB/s	26KB/s	1:0.3
4	80%	80KB/s	16KB/s	1:0.2
10	90%	80KB/s	8KB/s	1:0.1

As long as the attackers are using repeating ANY query attacks, RRL is a very effective solution[39] completely removing the amplification factor of the attack. However, the effectiveness of RRL decreases when the attacks get more sophisticated. Such as when the attacker uses a large set of DNS servers and distributes the queries over them not triggering the rate limiting factor and still achieving full or close to full amplification. Therefore, without extra measures, there is no easy way to completely stop DNS amplification attacks without several side effects until most ISPs make source address validation a default setting.

2.4.4 IPSec

IPSec[11] is a network protocol suite that authenticates and encrypts packets of data sent over a network. It guarantees data origin authentication through the packets Authentication Headers. However, it does not guarantee non-repudiation, which means that IP spoofing is still possible. To guarantee non-repudiation, a trusted third party is needed to certify each entities signatures.

A widespread implementation of IPSec that guarantees the senders authentication would imply that every Internet user would need to be granted a signed certificate by an authorized third-party entity. The sheer complexity of the processing required by the certification chains makes this approach not realistic.

2.4.5 HIP

Host Identity Protocol(HIP)[24] is an inter-networking architecture that enhances the original internet architecture by adding a name space between the IP layer and the transport layer. This new name space consists of cryptografic identifiers that will replace the role of IP addresses in naming application end-points(sockets). IPv4 and IPv6 addresses are still used, but only as names for topological locations in the network.

HIP delves into some of the problems of the contemporary Internet such as: loss of universal connectivity, poor support for mobility and multi-homing, unwanted traffic, and lack of authentication, privacy and accountability. Let's focus on the properties of HIP that directly impact the problem of the unwanted traffic, e.g. flooding or other malicious packets.

In the current message passing paradigm of the Internet, there is a sender and a receiver. The sender creates a message and sends it to a specific receiver by naming the message with a destination name. However, it's worth noting that in this paradigm, the sender has all the power, when a sender dispatches a message, the network itself has no idea whether or not the recipient of the message will be interested in it. Only when the message arrives at the recipient host is the consent consulted, and only then is the message dropped.

HIP offers two types of measures against unwanted traffic: making the recipient names not immediately accessible by the senders or by requiring the consent of the recipient before the networks delivers him any packet; also, with HIP it's possible to raise the cost of sending packets or reduce the cost of receiving them[15].

2.4.6 EIP

Ephemeral IP[19] is a preliminary proposal that aims to guarantee the integrity of source IP addresses. This proposal is based on LISP[5] and proposes the utilization of cryptographically generated IP addresses to be used as contextual identifiers, and, when accompanied with a self-generated certificate, allow for the verification of the source

identification of a packet. In order to guarantee the integrity of the tunnels source locator[9], it's possible to optionally trigger a cryptographic challenge to the sender, which if responded correctly guarantees the integrity of both the senders identity and it's locator.

When using challenge based handshakes in a TCP connection, EIP guarantees that all types of SYN flood attacks are nullified. EIP also makes it harder for an attacker to exchange packets with other hosts, effectively also making harder the task of gathering reflectors for a reflection based DDoS attack.

As presented in [19], EIP is able to drastically reduce the power of some of the most deadly attack vectors. But it has a few glaring flaws that we will discuss in the next chapter.

2.5 Attack vector trends and comparison

In this section we will analyze and compare the different studied attack vectors. We will take in consideration popularity, effectiveness, type of resource depleted, layer of the attack and the necessity of a spoofed IP.

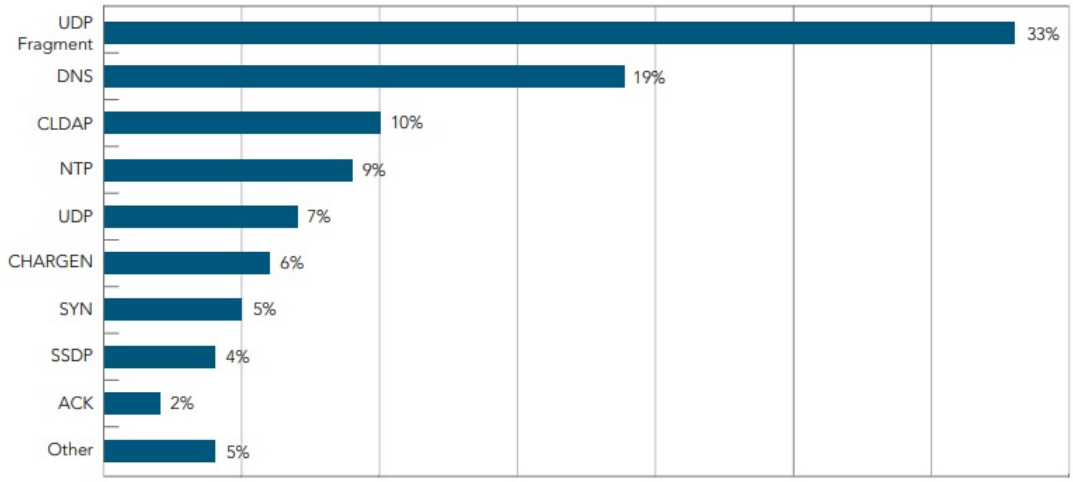

Figure 2.6: DDoS Attack Vector Popularity in Q4 2017, as reported by Akamai in [27].

As seen in the former sections, there are many ways to inflict a DDoS attack. There are many factors that may contribute to a certain attack vector being popular, like type of zombie needed and available exposed reflectors but one trend is certain, the attackers seem to favor either very simple attacks like UDP Fragment, UDP Flood and SYN Flood, or attacks with a high amplification factor like DNS Reflection and NTP. In Figure 2.7 we can see the DDoS Attack trends for the Q1 of 2017, provided by the Q1 2017 Akamai security report[27].

In table 2.4 we present a general overview of the discussed attack vectors, as well as a popularity and efficiency comparison.

[9]LISP is based on the idea that the Internet architecture is combined into: locators(where a client is attached to the network) and identifiers(who the client is)

Table 2.4: Attack vectors comparison

Attack Type	Main resource depleted	Layer	IP Spoofed	Popularity	Efficiency
DNS	network	application	yes	*****	*****
CharGen	network	application	yes	****	****
NTP	network	application	yes	****	****
SSDP	network	application	yes	***	****
CLDAP	network	application	yes	**	****
Ack Storm	network	transport	yes	*	**
Opt. Ack	network	transport	no	*	***
UDP Frag	processing capabilities	transport	yes	*****	*****
Syn Flood	processing capabilities	transport	yes	**	****
Direct UDP	network	transport	no	***	*
HTTP Flood	network	application	no	**	**

Breakdown of Table 2.3 The columns Attack type, Main resource depleted, Layer and IP Spoofed are self explanatory and Popularity is derived from Figure 2.7. We are left with Efficiency, this metric tries to quantify the amount of effort the attacker needs in order have the highest amount of impact in its DDoS attack vector. Not to be confused with the amplification factor, the latter only takes into account the size ratio of request/reply messages. With efficiency we take into account:

Method complexity The implementation complexity of the attack vector.

Method requirements The requirements needed to execute, i.e network knowledge, connection parameter knowledge, etc.

Zombie type needed Can it be done with a user level zombie, or does it need root level access.

Amplification factor Amplification factor is also taken into account.

Area of effect Does it effect a single server or does it also affect the surrounding network and the path between the attacker and the victim.

2.6 Summary

In this chapter we have introduced the necessary literature for the understanding of the next chapters. We have seen that different types of attack vectors require different resources from the attacker, e.g. reflection based attack require root-level access zombies and Ack based attacks require knowledge of the network.

The currently available defense mechanisms are limited in their effectiveness due to the lack of incentives for the ISPs (Internet Service Providers) to mediate and authenticate

the source addresses in their network, effectively heavily limiting spoofing based attack vectors.

Also, in the Internet, everyone can send packets and the willingness of the destination to received them is not controllable.

We have seen that the Internet of Things already has a huge impact on the biggest DDoS attacks today, and it is predicted that the number of these devices will continue to grow, so will the intensity and frequency of large scale DDoS attacks. Namely attack vectors that do not require IP spoofing, as with the increasingly availability of vulnerable machines, the effort required by the attacker decreases, as he may only need connection based attack vectors such as HTTPS flooding to cause a large impact on its victim. Such an attack requires only user-level access zombies which are easier to acquire.

Chapter 3

End-to-end Network Layer Security with HIP

In the last chapter we have seen several methods of DDoS defence. However, most of these methods require important changes in the Internet structure, or lack incentives for their adoption (e.g. ISP anti-spoofing practices). The most common and successful ones require costly infrastructures, e. g. specialized Cloud services. None are end-to-end or do allow progressive adoption without heavy investments.

A potentially better method would consist on a "magic-filter", implemented in an end-to-end fashion on the communicating entities or near them, in the Internet's edge routers. This solution would assure that all data traffic that went through the filter would be composed of: traffic intended to establish a data flux between two parties, where the receiving end has accepted the exchange; traffic related to a data exchange of a previously mutually agreed interaction.

However, this type of solution would still be vulnerable to a flood of message exchange requests. To mitigate it, filtering measures would need to be implemented near the source, through limiting the number of new message fluxes a party can try to establish in a given time interval.

Thus, a solution to this problem would be to have a generalized adoption of IPSec. Though, this solution is complex and lacks incentives for its adoption since at the transport level there are several popular security alternatives (e.g. TLS, HTTPS, SSH).

As we have also seen in that chapter, a proposal to boost the usage of IPSec in a host to host manner in a more realistic way, is HIP (Host Identity Protocol). This protocol facilitates the establishment of authenticated host-to-host network level channels. Its generalized adoption would be of great help to combat DDoS.

Unfortunately, HIP has not been widely adopted. Probably because it also contains some adoption barriers and lacks the right incentives. However, we think that if most of

these barriers could be smoothed, or removed, a solution with its advantages but without its drawback or barriers for adoption, could be of great help.

In this chapter we will delve deeper into HIP in order to get a better understanding of the reasons that led to its poor adoption. Then, we will try to devise ways to overcome these adoption barriers, while still allowing it to fulfil all its promises as an end-to-end security solution able to also combat DDoS.

In section 3.1 we analyse the HIP protocol in a more in depth approach.

In section 3.2 we analyse the possible DDoS attacks based on HIP itself and how HIP tries to overcome them.

In section 3.3 we discuss the reasons that led HIP to have very low adoption as it's only applied in a few private networks.

In section 3.4 we present three HIP protocol variants and discuss their motivations and desired impact.

In section 6.5 we conclude the chapter.

3.1 HIP Description

The Host Identity Protocol is a networking architecture developed by Ericsson and Boeing, among other companies, and academic institutions, since 1999. It has been standardized by IETF[23].

The original ideas were formed from the need to devise a new architecture that would solve some of the most challenging problems in the contemporary Internet: loss of universal connectivity, poor support for mobility and multi-homing, unwanted traffic, and the lack of authentication, privacy and accountability.

HIP is an implementation of the identifier/locator split[35] approach in the stack. That is, while in the current IP architecture, IP addresses take both the role of the host identifier and the host locator, in HIP these roles are separated, Host Identifiers take the role of Identifiers and the traditional IP addresses take the role of locators.

As a result of this split, the way applications send packets in this network is very different, since instead of referring to IP addresses they refer to Host Identities (HI), i.e., public keys. In HIP the Host Identity is a public key of an asymmetric key pair. Public keys are large in size and impractical to use as identifiers in the message exchanges. Thus, the need for a Host Identity Tag (HIT), which is an hashed encoding of the HI, used to represent the HI and has the following three properties: i) It's the same length as an IPv6 address; ii) it's self-certifying (i.e., given a HIT, it is computationally hard to find a HI key that matches the HIT); and iii) the probability of a HIT collision is extremely low.

3.1.1 HIP goals and solutions

As previously mentioned, HIP tackles some of the most challenging problems in today's Internet.

Loss of universal connectivity Loss of connectivity is caused by NATs, firewalls and dynamic IP addresses. While this may not seem a problem because of how familiar it is, originally the Internet did have a universal naming scheme. HIP achieves this with it's Host Identifiers that allow the IP addresses to be used as ephemeral locators, along side a number of design details related to middle boxes, such as the ability for hosts to directly authenticate themselves to firewalls via explicit registration.

Poor support for mobility and multi-homing In order to map mobile entities identities to their dynamic changing locations, a new level of indirection not present in typical DNS mappings is needed. The cryptographic nature of the Host Identifiers and them serving as a new name space creates the needed level of indirection that provides HIP with a new approach to implement mobility and multi-homing. In most identifier/locator schemes, this service is implemented as a mapping service[5]. HIP proposes the use of the DNS for this purpose.

Unwanted traffic Spam, phishing and DDoS are an everyday problem to every ISP or service. The current Internet architecture is an extension to the message passing paradigm, a sending process creates a message and sends it to another process by naming the recipient with a name. The problem is that all the power is given to the sender, as the recipient's address can be easily obtained and there are no incentives to refrain from sending additional messages, since sending for any reason has little to no cost. Thus the existence of spam and most DDoS attacks.

Lack of authentication, privacy and accountability Today, the authentication problem is technically easy to solve, but socially very challenging to adopt. Thus, HIP threats this problem in a per case basis between the end-users. HIP does not directly provide a solution to the privacy and accountability problems, but the use of cryptographic host identifiers to support the establishment of connection paths makes it easier to attribute ownership of a certain key to a distinct host. Also, the separation of Identifiers and Locators makes it easier to hide the topological location of communicating parties.

3.1.2 How HIP works: Base Exchange

The HIP base exchange, from now on referred as BEX, is used to establish a pair of IPSec security associations (SA) between two hosts. It's built around an authenticated Diffie-Hellman key exchange[28] but with some unusual features related to DDoS-Protection.

As seen in Figure 3.1, the introduction of the new Host Identity layer means that communicating hosts don't need to know their counterpart's locators, only their Host Identity. The Host Identity layer maintains mappings between identities and locators. Thus, when a mobile host changes its IP address, HIP is used to transfer that information to all peer hosts. Upper layers, e.g. applications, remain unaware of this changes.

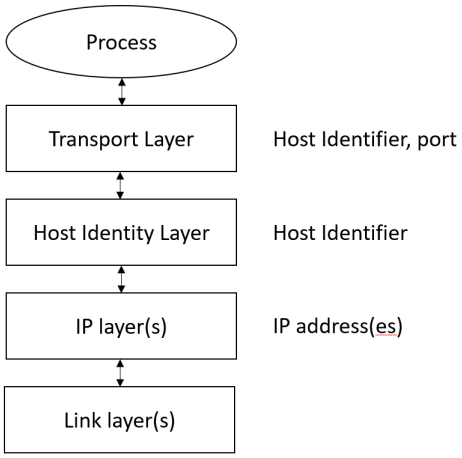

Figure 3.1: Approximate location of the HIP sublayer within the TCP/IP stack.

The HIP BEX consists of a two round handshake, in which the hosts exchange Diffie-Hellman public values and create a shared Diffie-Hellman key. This key will be used to generate keying material for several cryptographic operations, such as message integrity and confidentiality. Currently, the default option is to used them to establish a pair of IPSec Encapsulated Security Payload (ESP) Security Associations (SA) between the hosts.

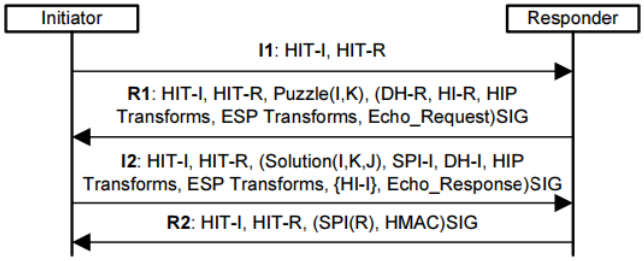

Figure 3.2: HIP base exchange messages.

Figure **3.2** depicts the HIP BEX, we shall now delve further on each separate message.

I1 The Initiator receives the Responder's HIT-R (Responder's HIT) from either a DNS lookup, some other repository, a mapping service or a local table. Since I1 is so easy to spoof, no attempt is made to add to its generation or processing cost.

R1 The Initiator's HIT-I must match the one received in the I1 packet. Puzzle(I,K) contains a random I number and a difficulty level K, we shall see a more in depth overview of the puzzle mechanism in a later subsection. The parameters listed so far are the non signed part of R1. In order to reduce the overhead of generating Diffie-Hellman values, the R1 packet is pre-signed, the signed values are the Responders Diffie-Hellman value (DH-R), it's Host Identifier (HI-R), several transforms related to cryptographic suite preferences (HIP Transforms, ESP Transforms) and the Echo Request. This last parameter takes an important role, in it the Responder can remain stateless during this part of the BEX. The responder ciphers an

hash of all the needed information related to the puzzle mechanism and forces the Initiator to return it unchanged in the next I2 message, this way the Responder can verify that it indeed sent the corresponding R1 and that the puzzle solution in I2 follows the given parameters.

I2 In this packet the Initiator sends the puzzle solution (Solution(I,K,J)), it's Security Parameter Index (SPI-I) in order to later create the IPSec Security Associations, it also sends its Diffie-Hellman public value (DH-I) and encrypted Host Identity (HI-I), along with the cryptographic protocol transforms (HIP Transform, ESP Transforms). As referred earlier, the Initiator needs to send Echo Response as it was sent from the Responder, untouched. The Initiators Host Identity is encrypted with keying material derived from the Diffie-Hellman values. The whole message is signed, with the exception of both parties Host Identity Tags (HIT).

R2 The final message in the HIP BEX serves as a way to confirm that the Diffie-Hellman shared secret key was successfully generated by both parties, besides both entities HIT values (HIT-I, HIT-R), the Responders Security Parameter Index value (SPI-R) is also included. A HMAC is calculated over the whole HIP packet with the Responder's HI concatenated to it. The message is then signed.

3.1.3 Puzzle mechanism

The purpose of the HIP puzzle mechanism is to protect the Responder from DDoS threats. It allows the Responder to delay state creation until the Initiator has proven to be "sincere" in the sense that it used enough CPU cycles to solve the puzzle.

The puzzle is based on finding a value J, such that the hash of the concatenation between J, I and both parties HIT values has K least significant bits set to 0. To solve this, the Initiator must compute the solution using a brute force approach as the Responder selects an I value that cannot be previously guessed by the Initiator. The Responder can, this way, set the level of trust on the Initiator in the form of the value of K. It can also alter K based on it's own load level.

3.2 HIP Attacks

When HIP BEX ends, a secure and authenticated channel is created between the end-users since both parties know the necessary Security Associations and a shared key. This grants a robust defense mechanism against DDoS but introduces significant overhead. Additionally the BEX is itself susceptible to attacks based on its message exchanges [23, 24].

Replaying Message replaying can be done in several phases of the HIP BEX, thus HIP has different ways to deal with this attack. Responders are protected against replays of I1 packets by virtue of the stateless response to I1 with pre-signed R1 messages

and Response Rate Limit mechanisms[39]. Initiators are protected against R1 replays by a monotonically increasing R1 generation counter parameter included in R1. Responders are protected against replays of forged I2 messages by the puzzle mechanism and the Echo Request field.

I1 Flood The purpose of an I1 flood is to try and deplete the available pre-signed R1 messages of the responder. HIP tries to solve this by attributing the same Diffie-Hellman values to different I1 messages, however this introduces a statistical probability of Diffie-Hellman value collision.

I1 Spoofing An attacker by spoofing a I1 message with another user HIT can try to make an unsuspecting user of receiving and possibly solving a puzzle he hasn't requested. HIP solves this by discarding all R1 messages where an I1 message wasn't first sent by use the HIP state machine.

I1 to R1 Reflection attack Related to the previous attack, I1 to R1 Reflection attack uses spoofing to get unwanted traffic to a victim. As seen earlier, spoofing gets the victim to discard the messages, but the messages still arrive to the victim. Since the size difference of I1 and R1 packets is very significant, an amplified reflection attack is possible in this round of the BEX. HIP uses as a solution a mechanism studied earlier in Chapter 2, response rate-limiting of the sent R1 by the IP address parameter. With a fined tuned value for the RRL it's possible to severely mitigate reflection attacks this way.

I2 Flood In this attack, the attacker correctly solves the puzzle on R1 but then spoofs its IP address and uses an invalid or someone else's HIT in order to try to make the responder, upon seeing a correct puzzle solution, computes shared Diffie-Hellman secret keys for various spoofed entities. HIP solves this by, after an N number of bad I2 messages containing a certain solution, all following I2 messages with that solution are discarded.

3.3 HIP Adoption barriers

So far, HIP seems like a worthwhile upgrade to the current Internet so why hasn't it had a more widespread implementation? The answer to this question is that HIP faces several techno-economic adoption barriers caused by HIP complete network architecture turnaround. The Internet has been based on the TCP/IP stack since it's inception and the changes HIP requires are simply too many and too severe.

In 2013 a survey was made on the reasons for the non-deployment of HIP, the interviewees' consisted of stakeholders in operating system vendors, Virtual Private Network (VPN) software providers, mobility, security and sensor solutions, Internet service providers and professionals with expertise in the networking and data security. Seen

#	Reason for non-deployment	1	2	3	4	5	NA
1	HIP is missing a killer application.	0	0	4	7	5	0
2	Substitute technologies (possibly already been deployed) are favored.	1	1	3	9	2	0
3	Chicken-egg problem (i.e., no one adopts HIP because early adopters do not gain immediate benefit).	2	2	4	4	3	1
4	HIP is a too big change and people favor point solutions solving a single problem.	1	5	3	3	4	0
5	Lack of stable mainstream implementation discourages adoption.	2	3	3	4	3	1
6	Real-world deployability not taken into account from the first day on due to research-mindedness.	4	1	4	6	0	1
7	There is no real demand for HIP.	5	2	2	4	3	0
8	Decision-makers are not aware of HIP or they have misinformation.	5	2	4	1	3	1
9	Company policies or personal frictions in the IETF hinder deployment.	6	2	4	2	2	0
10	Experimental track status discourages adoption.	7	2	4	2	1	0
11	Managing identities is too problematic.	7	2	1	3	0	3
12	HIP is too late.	7	3	0	3	0	3
13	ISPs (or other stakeholders) do not like encrypted traffic.	9	4	1	1	1	0
14	HIP is on the wrong layer of the protocol stack.	10	2	1	1	1	1

Figure 3.3: Reasons for HIP non-deployement[15].

in Figure 3.3 is the ranked list of these reasons. These can be grouped into 5 main HIP adoption barriers, in order of importance:

Demand is low Demand for HIP functionalities is low, and where demand exists, substitutes have been favoured either because they were earlier in the market or because they have relative advantages due to some design choices of HIP. Additionally, HIP solves problems that may not be demanded by applications, e.g. seamless mobility that preserves TCP connections may not be needed, as a majority of network traffic is HTTP according to[18].

Substitutes arrived earlier HIP was introduced considerably later than its core substitutes IKE[9] a protocol used to set security associations, Mobile IP[25] which allows the usage of the same IP even when changing networks and TLS, a set of cryptographic protocols used to provide privacy and data integrity. These substitutes solutions were already accepted by the standardization bodies and the industry before HIP even started its development. This made HIP lose the headlines and decreased the general public's knowledge. These substitutes also evolved in parallel to HIP, e.g. VPN providers that decided to add mobility to IKE, decided to specify MOBIKE[3] instead of changing to HIP.

Substitutes have advantages The lower complexity implementation of using problem-specific protocol solutions is favoured to the approach of HIPs complex, general solution. Additionally, problem-specific solutions are optimized in fixing a very specific problem, they need less patches and are thus easier to deploy.

Early adopter benefits requires costly coordination HIP requires both communicating partners to support HIP before either of them gets any benefits associated with HIP. This creates a problem for early adopters as they will only benefit after the critical mass has adopted HIP, obviously, stakeholders are not willing to invest in such changes unless they are certain that other stakeholders also adopt HIP.

Research-mindedness of HIP developers has lead to strategic mistakes HIP developers visualized HIP as a protocol that would change the whole Internet, which can be seen by outsiders as lack of realism. They focused too much on the Identifier-Locator split instead of its practical benefits, and their search for theoretic perfection has lead to non-optimal design choices from the deployment point of view.

3.4 HIP variants

The HIP BEX involves heavy cryptographic computations on both Initiator and Responder. The heaviest component is the computation of the Diffie-Hellman key, immediately followed by the computation of the Diffie-Hellman public values. Signature computations and verifications over messages R1, I2, R2 represent a lesser overall computational expenditure but are non negligible. From this observation, two modifications of HIP have been proposed in order to make the protocol used by resource constrained hosts. Another HIP variant is based on distributed load sharing[32]).

3.4.1 HIP Diet Exchange (DEX)

DEX[22] proposes that each host uses a long-term elliptic curve Diffie-Hellman[10] (ECDH) public value as its host identifier and adapts the key exchange as seen in Figure 3.4.

As the Host Identifier is itself also the Diffie-Hellman public value, there is no need to authenticate it through asymmetric cryptography. Knowledge of the DH key is enough to prove that a host is a legitimate peer in the exchange. Additionally, the DH key is also used to transport two random seeds x and y that will be used to derive the final shared key.

The static nature of the ECDH eliminates the recurrent DH key generation costs, and make the key exchange lighter. However, this was still though to be too heavy for resource-constrained hosts, so an even lighter variant was devised.

3.4.2 Lightweight HIP (LHIP)

LHIP is a more radical approach, it uses the same syntax as in the original HIP BEX but doesn't use any of its security mechanisms in order to achieve the lightest HIP variant with the least amount of compatibility issues. Therefore, no Diffie-Hellman key is computed, no RSA operations are performed and thus, no IPsec tunnel is set after the exchange.

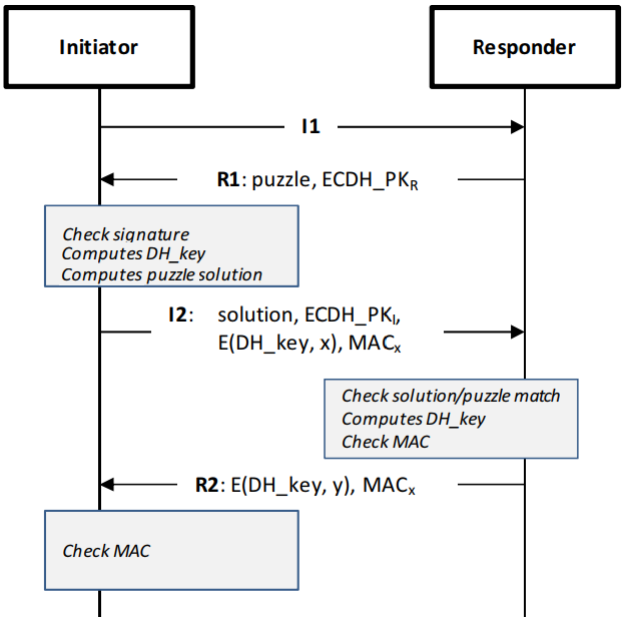

Figure 3.4: HIP Diet Exchange.

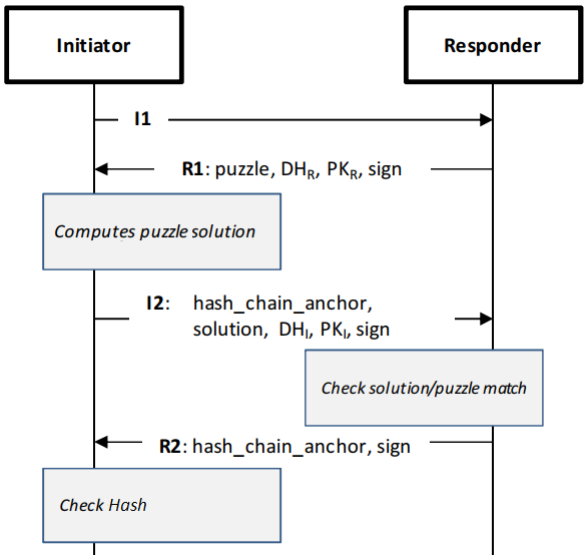

Figure 3.5: Lightweight HIP exchange (LHIP).

Instead hash chains are used to cryptographically bind successive messages with each other, which represents a minimal degree of security. LHIP exchange is depicted in figure 3.5.

It's worth noting that even though the fields representing the Diffie-Hellman Responder and Initiator values (DH-R and DH-I) and their public keys (PK-R and PK-I) are present, they are unused in the LHIP exchange and are only present for compatibility reasons and because LHIP supports the upgrade to standard HIP BEX.

LHIP does not provide strong host authentication, the hash chains only guarantee that the ongoing session has not been hijacked. Therefore LHIP trades security for energy

efficiency in a drastic manner.

3.4.3 HIP Tiny Exchange (TEX)

TEX aims at reducing the computational requirements of running HIP BEX while still retaining its basic features, like using the keying material obtained from the transactions to create a post-TEX security mechanism such as IPsec.

HIP TEX is based on two design decisions:

Replacement of DH with Public Key Cryptography The computational cost of a key exchange using Diffie-Hellman values (one DH exchange + one signature/verification) is around 2.5 times higher[32] than using a 1024-bit RSA algorithm (one encryption/decryption + one signature/verification). On this premise, TEX puts aside Diffie-Hellman calculations in favour of computationally less expensive approach that still ensures secure HIP connections. This changes still weren't enough for a highly constrained host, therefore TEX presents another key feature.

Collaborative key exchange approach In a scenario where a highly-constrained host node is surrounded by possibly trustworthy peer nodes, TEX proposes to delegate heavy computational load from public key cryptography to less constrained nodes in its neighbourhood.

During the key exchange, these assisting nodes, called "proxies", take charge of secret key delivery using asymmetric cryptography primitives in a distributed and collaborative manner. Each proxy encrypts a different part of the secret key sent by the highly resource-constrained Initiator to the Responder. Proxies also have to receive the secret key generated by the responder in order to decrypt it and transmit it to the Initiator. The reconstruction of the session key is completed by both the Initiator and the Responder in order to guarantee its secrecy.

3.5 Conclusion

In conclusion, the key features of HIP impose several changes that led to the protocol's adoption barriers. Moreover, HIP functionality requires a heavy BEX with two rounds which introduces significant overhead. In certain scenarios this overhead would be bigger in both size and computational requirements than the expected end-to-end transaction, e.g. DNS and other Request/Reply protocols. This means that using HIP as a defence for DDoS attacks that involve this protocols is inefficient and other methods may be a better option. Another problem with HIP adoption occurs in a situation where security is preferentially deployed in a layer higher than HIP. Thus, using similar security mechanisms in different network layers would introduce redundancy but, e.g. by using tls without HIP, and knowing that TLS only starts after TCP, we would still have SYN flood type attacks.

Chapter 4

End to end Light Security Protocol

Our goal in this chapter is to present the End to end Light Security Protocol (ELSP). In order to create an end-to-end network layer protocol that complements IP with practices allowing the mitigation of DDoS attacks, one should focus on the good security practices carried out by HIP, such as its puzzle mechanism, and if required, the ways in which it deals with protocol attacks.

However, for a protocol with this purpose, we shall not worry about user authentication or any other sub-problem that HIP tries to fix that isn't directly linked with DDoS defense. Other examples are the problem of loss of universal connectivity, support for mobility and multi-homing and the lack of privacy and accountability. Thus, ELSP should not need any major change to the Internet's architecture, should not need third party entities to account for user authentication, as authentication per se, isn't a DDoS problem.

This protocol adopts features from HIP to a certain degree. In fact, if HIP was widely adopted, since it offers a solution based on strong and proven defence mechanisms, DDoS at network and transport level would be more easily controlled and it would be easier to find tools to combat it at other levels (e.g. application). Unfortunately, this is not the case, and, as we have seen in the previous chapter, it probably won't ever be.

As seen in the previous chapter, some HIP features seem hard to justify in a DDoS mitigation context or were clearly an over-specification in the sense that the adoption of certain mechanisms are mandatory even when they are useless (e.g. Diffie-Hellman key exchange in a scenario where a connectionless approach would be desired).

However, if we build on the HIP proposal and refrain from adopting its most arguable features that lead to its adoption barriers, it may be possible to devise a solution that still retains the end-to-end DDoS mitigation features found on HIP.

That is the goal of ELSP, which we will present bellow.

In section 4.1 we introduce ELSP and give an overview of the protocol functionalities and how it tackles the DDoS problem.

In section 4.2 we present how ELSP behaves when in a connectionless scenario, based on UDP-like interactions.

In section 4.3 we present how ELSP behaves when in a scenario where the Initiator intends to establish a non-secure exchange with the Responder.

Finally, in section 4.5, we analyse the security properties of ELSP and how they can be compared to the guarantees that HIP BEX provides, by discussing both protocols behaviours when faced with the same attack vectors.

4.1 Overview

Even though HIP faces many fierce adoption barriers that prevents it from being deployed solution, it may be possible to create a protocol prototype inspired from it, where we can remove the barriers that contributed to its non adoption while retaining its security features as much as possible. In order to do so, we introduced the following modifications:

Public keys are not to be used as identifiers and their certification is orthogonal to the protocol This means that in ELSP, public/private keys are just that, and ELSP doesn't rely on any particular certification method.

Thus, there are no certificate exchanges in the protocol And if keys need to be certified in a certain context, any suitable mechanism can used.

Public keys are to be exchanged directly As it is in SSH and TLS to avoid the need of third party key distributors.

Exclusion of Identifiers The packets exchanged under ELSP are not tagged by identifiers, but are tagged by message exchange nonces. To be able to invert, without previously knowing such nonce, is very hard and would require an enormous number of probes.

Both parties should only pay what it uses This means that ELSP should allow the Responder to fine tune the level of security and complexity of the mechanism it requires when exchanging packets with the Initiator, this also means that when the goal is to use security at the transport level, it is useless to pay the price of redundant features at lower levels. Thus, ELSP strongly adheres to the end-to-end principles, which HIP didn't.

Early in the development of ELSP, we have found that in order to efficiently mitigate DDoS attacks by following HIP's security mechanisms, and at the same time refraining from having the same adoption barriers, we had to treat different scenarios in a flexible way. An Initiator that intends to only complete a Request/Reply round should be treated

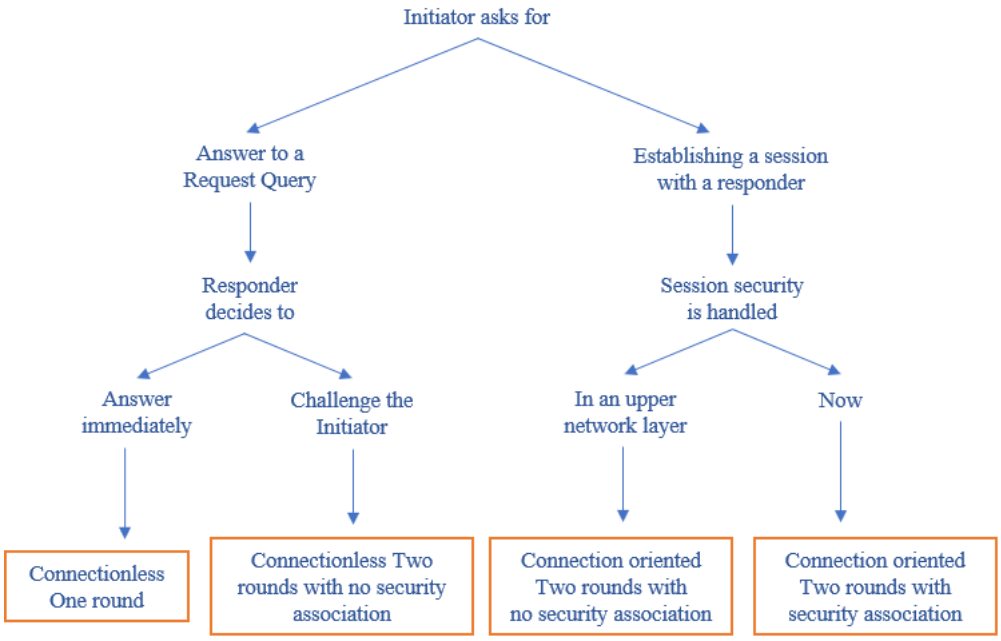

Figure 4.1: ELSP varied messages exchange flows tailored for each specific situation.

differently from an Initiator that intends to establish a long term TCP connection. Also, an Initiator that pretends to establish a session relying on security mechanisms in another layer, should be able to bypass all redundant lower level security mechanisms if it so desires.

Therefore, we have defined ELSP from the ground up with this in mind. Figure 4.1 further illustrates the reasoning for the different message exchanges that represent the different scenarios that will be developed in the next sections.

Furthermore, for a better understanding of the capabilities of ELSP, we need to make clear the type of attacks that it's designed to mitigate and which type of attacks it has no intention of mitigation. Firstly, the protocol is designed to mitigate DDoS attacks, these attacks are usually based on either depleting the computation resources of a victim or exhausting its bandwidth.

With the former in mind, the attacker's model for ELSP doesn't contemplate attack types such as:

Eavesdropping In a non secure, "cleartext", network communication, an attacker that has access to the victim's network can read the ongoing message exchange and thus, access potentially sensitive information. In order to effectively mitigate this type of attack, strong encryption and a certified key distribution mechanism is required. In the perspective of DDoS mitigation, having these security mechanisms mandatory would introduce significant overhead when they are not needed. Thus, in ELSP we make use of these mechanisms optional.

Man in the Middle These attacks differ from the previous Eavesdropping attack in the fact that the attacker not only reads the messages but also changes them in order to assume someone else's identity to harm the victim. To mitigate these type of attacks, all communicating entities would need to have certified keys and, as discussed earlier, this overhead is taken as optional and thus ELSP doesn't necessarily mitigate MitM attacks per si.

As the purpose of this work is to mitigate DDoS attacks, we made the decision to keep using the already proven mechanisms when dealing with a similar to HIP variant. Thus, we did not explore the options studied earlier in Chapter 3, like the replacement of Diffie-Hellman with RSA public key cryptography.

Having made clear the purpose of ELSP we shall now delve into the discussion of its variants.

4.2 Connectionless One round Variant

One of HIP's biggest adoption barriers was the fact that according to some sources[18], nearly 60% of the Internet's interactions are simple Request/Reply messages, and many of them connectionless. In a normal HIP environment in order for the Initiator to receive the reply to its request, 2 obligatory security rounds need to first take place and then another with the request and reply exchange. Thus, we devised a message exchange that only needs one round in a typical, best case connectionless scenario, while still retaining optional tools to mitigate DDoS attacks.

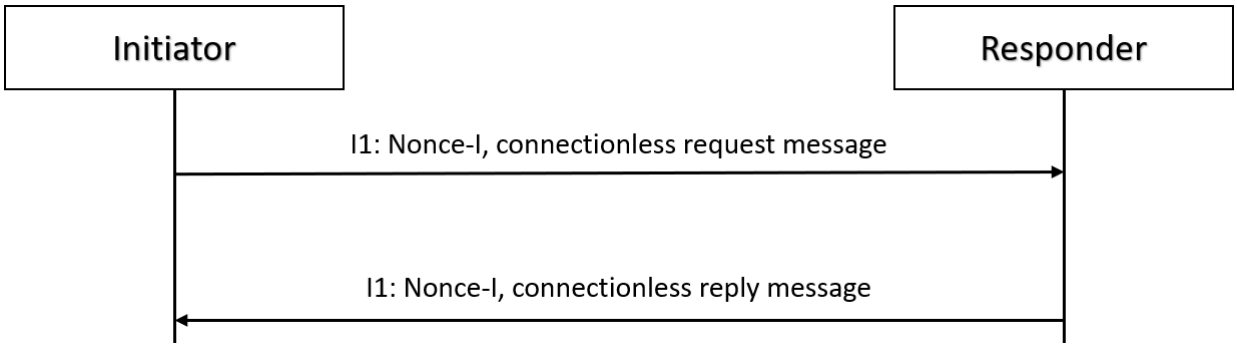

Figure 4.2: ELSP One round connectionless scenario message exchange.

As seen in Figure 4.2, base exchange in this scenario is very simple. The I1 message only contains a random generated number Nonce-I and the request itself, the R1 message correspondingly returns Nonce-I and the response.

The best case scenario is based on the Responder's acceptance of the I1 message, which is determined by the evaluation of three major metrics: the Responder's current load values, the amplification factor correspondent to the query/response at stake and

whether or not the Initiator's IP address has been flagged by the Responder's Response-Rate-Limiting (RRL) mechanism. If the Responder decides that a certain I1 message has a low probability of being part of a DDoS attack, e.g. , the Responder receives an UDP query with a low amplification factor, with an IP address that hasn't been flagged as malicious by the RRL mechanism, and at that moment, the Responder is under low load. It may then decide to answer immediately.

The types of DDoS attack vectors that use typical UDP Request/Reply messages are, as we have seen in Chapter 2, amplified reflection attacks and flooding attacks. This message exchange nonce's in conjunction with a RRL mechanism on the Responder side provide for a meaningful defence strategy against reflection attacks.

In a scenario where this message exchange is implemented, any receiving party can very easily identify if the message received was actually a response to a previous query, by verifying if the current protocol state matches I1-Sent(Nonce-I). This prevents an entity from uselessly computing a received message that it did not request. However this doesn't prevent its network channels from being overwhelmed by bandwidth exhausting attacks such as a reflection attack. To make the attack less effective, RRL can be used on the Responder's side, the utilization of RRL and the assessment of its effectiveness shall be further discussed in a later section.

By now, we have shown how ELSP deals with a traditional connectionless scenario where only nonces and RRL mechanisms seem to be sufficient to mitigate Reflection DDoS attacks, the type of attack vector most prominent in this scenario.

4.3 Two round with no Security Association Variants

If the Responder is currently under heavy load, or because it has detected that the Initiator's IP address is linked to DDoS activities with a high degree of probability, the same message exchange listed earlier is not a sufficient solution.

In this scenario, we still want to resolve the connectionless scenario as efficiently as possibly while still mitigating possible DDoS Reflection attacks. We are now in the point where the Responder has received an I1 message and it has decided not to directly respond the Initiator. Thus, as we have learned from the defence mechanisms that HIP uses, we can make the Initiator prove that he has good intentions by the means of solving a cryptographic puzzle.

However, the message exchange can't continue as it does in HIP. In HIP the messages R1, I2 and R2 achieved DDoS defence by the means of its puzzle mechanism and achieved the creation of a future connection association by the means of exchanging Diffie-Hellman values and Security Parameter Index values.

The same applies in a scenario where session security is preferred to be handled in an upper layer of the network stack, where a security protocol may be obligatory and we want to avoid redundancy.

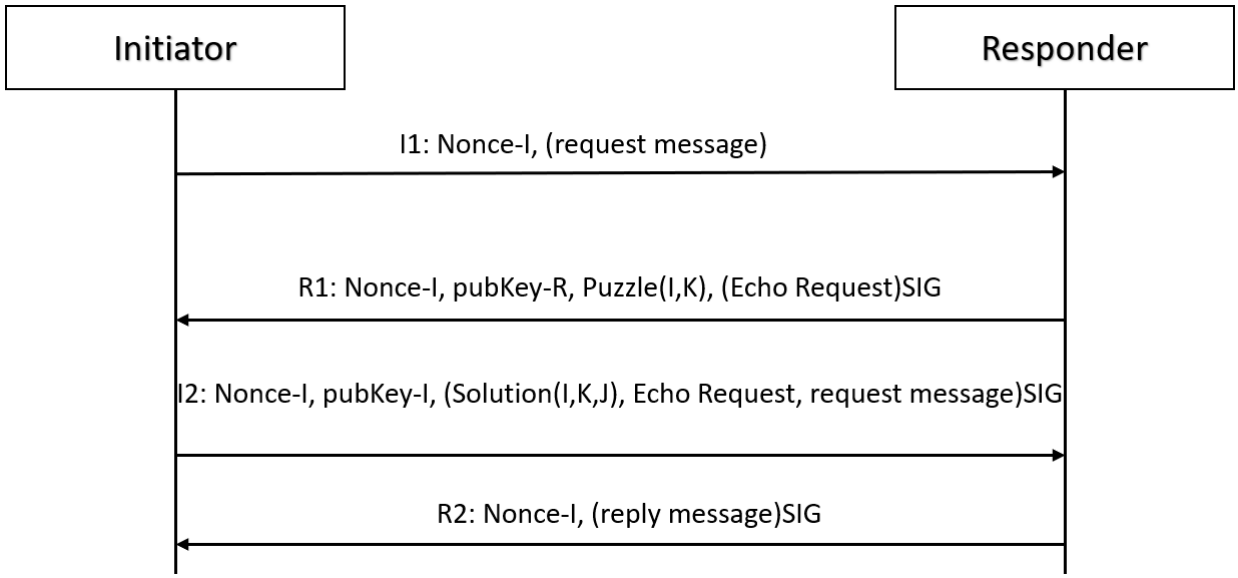

Figure 4.3: ELSP Two round with no session security variant message exchange.

As the current scenario seeks to solve DDoS attacks in a connectionless scenario, the security typically gained by the creation of an secure IPSec tunnel is useless and thus redundant. So, the message exchange in this scenario consists of:

R1 The Responder returns the Initiators Nonce-I and challenges it with a cryptographic Puzzle(I,K) where I is a random number and K is the difficulty level. In the Echo Request field we have an HMAC of the concatenation of the required information the Responder will need to check to prove the solutions authenticity. The responder's public key is also included in this message, in order to allow the Initiator to verify the given signature.

I2 The Initiator has to once again send the payload-request and the same Nonce-I value as the Responder doesn't keep any state of the message exchange. Solution(I,K,J) consists of the solution to the puzzle challenged in R1 where J is the solution and I and K are the numbers given in R1. In order to prove the solutions authenticity, the Initiator also returns the Echo Request. A signature of these parameters is also provided by the Initiator.

R2 The Responder starts by verifying the solution as it is the least computationally intensive task that can quickly discard false solving attempts. If the solution is correct, the Responder then verifies its authenticity by verifying the Echo Request HMAC. If sucessful, the Responder can now send an R2 message that consists of the Initiators Nonce-I and the payload-reply, followed by the Responder's signature of the reply.

In fact both connectionless variants are used in an integrated way. When the Responder receives the I1 message, it may reply in several different ways:

1. Discard the message.

2. Reply immediately.

3. Respond with a challenge.

This kind of mechanism is a good, application-independent generalization. The situation where the Responder answers with a challenge message is equivalent to current mechanism of DNS servers of responding to suspicious queries with truncated DNS responses, and if the querying party wants the rest of the Response, it will need to establish a TCP connection with the DNS server, thus mitigating possible DDoS Reflection attacks.

A connection-oriented version of this variant is based on the same principles. However, the I1 message does not contain a request message as the Initiator knows it will have to re-send it later on I2. Connection-oriented request message are e.g. the SYN message used in TCP.

4.4 Two round with Security Association Variant

The final message exchange variant is the closest to HIP, having both its puzzle mechanisms and computationally heavy shared key distribution with the Diffie-Hellman algorithm.

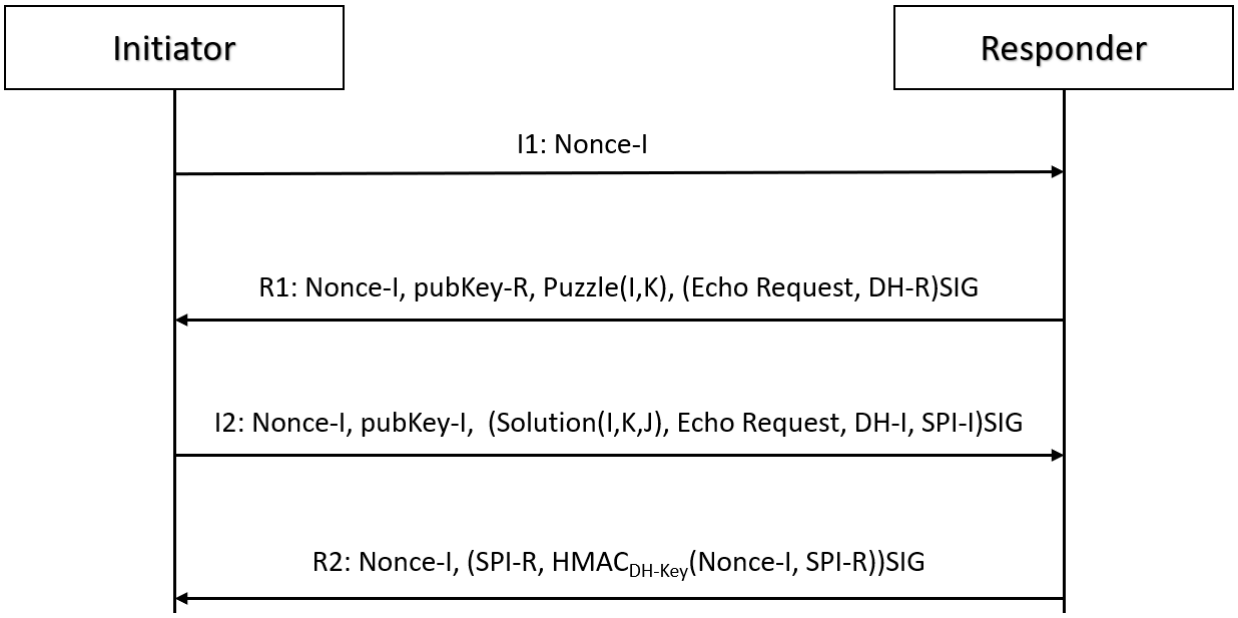

Figure 4.4: ELSP two round with session security variant message exchange.

I1 This message exchange begins with the Initiator sending an I1 message with the a on-the-fly, randomly generated value Nonce-I.

R1 The Responder, when receiving an I1 message, chooses a previously computed R1 message and adds the Puzzle I value, the Initiator nonce and the Echo Request field, these values are not pre-signed. The parameter that now differs from the message exchange listed on 4.3 is the Difie-Hellman value of the Responder (DH-R), needed to assure session security within ELSP. The last needed parameter on R1 is the Responder's public key (pubKey-R).

I2 The Initiator, when receiving an R1 message first checks if it is in state I1-Sent(Nonce-I), with a matching Nonce-I received on R1, then verifies the Responders signature. After these checks, the Initiator can now try to brute force a solution to the given puzzle and create its own Diffie-Hellman value. I2 is then composed of, Nonce-I and pubKey-I, as the Responder does not store any state on each of its ongoing message exchanges, the puzzle Solution(I,K,J) and the corresponding Echo Request, the Initiators public Diffie-Hellman value (DH-I) and the Initiator's proposed Security Parameter Index (SPI). Lastly, the Initiator produces a message signature covering the Solution(I,K,J), DH-I and the Echo Request.

R2 When receiving an I2 message, the Responder first verifies the puzzle solution as it is the least complex verification and immediately discards all I2 messages sent by entities that didn't solve the puzzle. Then, the Responder verifies the Echo Request and finally the Initiator's signature is checked. The R2 message is then composed by the Initiators Nonce-I, the Responders Security Parameter Index value (SPI-R) to later be used in IPSec. At this point, the Responder can successfully generate the shared Diffie-Hellman key and as a token of proof that both parties are in possession of the same Diffie-Hellman key, in R2, an HMAC of the Nonce-I and SPI-R values is included in the message. Lastly, the message is signed by the Responder. The Initiator, when receiving R2, generates the shared DIffie-Hellman key and verifies the received HMAC. All the components for the establishment of an IPSec secure tunnel are now available.

4.5 Preliminary Security Analysis

In this section we will discuss the security guarantees provided by ELSP in its variants. We will list the common attack vectors shared with HIP and what is the corresponding available solution in a case-by-case approach.

We can now discuss the main differences of the overall ELSP protocol with HIP, and how these differences may affect the effectiveness of ELSP when mitigating DDoS attacks:

- The security analysis done on HIP expects the usage of authenticated public keys by both parties, in HIP's case, these keys are obtained from third-party entities, making HIP a non end-to-end protocol. ELSP being an end-to-end protocol, keys are exchanged by both entities **when required**, thus avoiding redundant key exchange

mechanisms when not desired. However, as the authentication of keys on ELSP is mainly optional and we do not expect it to be used in normal situations, ELSP becomes vulnerable to Man in the Middle Attacks.

- The other main difference lies on the architectural decision of ELSP to not follow the Locator/Identifier separation paradigm. As so, ELSP does not use Host Identitifiers or Host Identity Tags, in its place, and only for tagging purposes i.e. not as identifiers, ELSP uses Nonce-I values. ELSP loses the ability to have self transforming identifiers, but as its purpose is DDoS mitigation, this does not affect its effectiveness.

Thus, we shall not analyse the HIP-like variant of ELSP, the two round with security association variant. As it was discussed in Section 3.2.

4.5.1 Attacks on the Connectionless One round variant

R1 Replay This attack is based on an eavesdropping entity storing R1 messages from a certain Responder to later send to unsuspecting victims and entice them to solve the cryptographic puzzle worthlessly.

ELSP solution: This is attack is fully mitigated by the inclusion of Nonce-I parameters on all I1 messages. If an entity receives an R1 or R2 message without being in the state I1-Sent(Nonce-I) or I2-Sent(Nonce-I), the message is immediately discarded.

I1->R1 Reflection An attacker spoofs the IP address of the victim and floods a typical Request/Reply server, the victim will then receive answers to queries it did not request.

ELSP solution: ELSP tackles this attack with two different mechanisms, first, having the RRL mechanism on the Responder's side set to not answer queries with a high amplification factor, e.g. DNS any-queries. Secondly, if an entity has a firewall that intercepts traffic before it reaches the main host, this firewall can now filter all messages with unrequested Nonce-I values.

4.5.2 Attacks on the two round no session security variants

I1 Flood This attack is based on the fact that the I1 message is very small and easy to create, on the other side the R1 message has computationally complex mechanisms like signature signing. This disparity makes I1 Flooding an effective attack vector.

ELSP solution: The solution is to have pre-signed stock R1 messages, that only need to be added exchange specific parameters. However this solution may not be enough as an attacker can try to exhaust all the available stock R1 messages of the Responder, the solution adopted for this problem is the same as used in HIP, when in shortage

of stock R1 message, re utilize the available R1 messages to different Initiators in a randomly distributed way. This can cause key collisions, and thus, a high stock of R1 is recommended.

R1 Replay This attack and its solutions are the same as described in the earlier subsection.

I1-R1 Reflection This works in the same manner as the one described in the earlier subsection, with the difference that in this message exchange variant, the R1 message is much larger, thus creating a larger amplification factor.

ELSP solution: The same base solution mechanism is the same as in the earlier variant, with the difference in the RRL parameter settings. The sent R1 messages from this variant have a different IP address bucket, with a stricter threshold.

Incorrect query I2 Flood This attack is based on the attacker correctly solving the received puzzle on R1, and then flooding the Responder with I2 messages that have both correct puzzle solution and correct Echo Request values but an incorrect or invalid query. The Responder will then exhaust its resources by continuously verifying the same puzzle response data.

ELSP solution: The solution for this attack is to store state when this occurs. When a certain number of times an I2 message has its puzzle solution verified but no R2 message is produced, that puzzle solution is blacklisted. The fact that the Responder needs to store state is offset by the fact that the attacker has to solve the puzzle in order to continue with the attack.

4.6 Conclusion

As many of the security mechanisms used in ELSP are the ones used in HIP, we need to correctly analyse the transition of these HIP functionalities to the normal Internet architecture. We need to understand what is being lost and how this affects the security guarantees of ELSP.

Routing is no longer based on Identifier Tags as the Identifier/Locator separation paradigm is abandoned, IP addresses will now take the role of both Identifier and Locator of an host. This removes the inherent HIP solution to the mobility problem, and more importantly, entities are no longer able to reconstruct another entities HIT values using its public key (HI) as we will no longer support the equivalent to Host Identity Tags. However this functionality of HIP only makes sense when Host Identities are certified by a third party, and since we leave this feature open to implementation, as HIP does, no real difference is to be had.

Regarding the different attacks on both HIP and ELSP we can conclude that ELSP doesn't weaken in any way the security features adopted by HIP.

Chapter 5

Quantitative Effectiveness Analysis

Obtaining attack traces that are simultaneously recent, publicly available, and representative of complete attacks isn't an easy task. Nevertheless, there are some sites that contain some attack traces[1,2]. Also, in a typical real scenario high scale attack, the attackers employ several different vectors, making it difficult to measure the effectiveness of a given protocol.

Analysing a DDoS mitigating protocol is tricky, even with access to concrete DDoS attack data, the analysis of the effectiveness of a given protocol is dependable on each attack situation, the same can also be said for simulated attack data. DDoS defence is a game of cat and mouse, where new vulnerabilities are always trying to be found by attackers and thus, they are always one step ahead. This leads to the conclusion that using concrete data to discuss effectiveness analysis is never enough. In fact, proving that a mechanism prevents an attack using any attack traces is like proving that a program has no bugs by doing some tests.

Protocols at the network and transport layer, can be exploited by as many attacks as there are different mechanisms within these protocols (e.g. SYN Flood, ACK Storm, Optimal Ack Attack, UDP Fragmentation).

In reality, one way to simplify such an analysis would be to establish a common barrier to all protocols with the usage of a unique initial message exchange mechanism, that once complete, would isolate all possible exploits.

For instance, if all traffic acknowledged by a party is composed of either the preliminary security message exchange and the then trust-secured message flow, then all the traffic that is needed to analyse is:

- Traffic needed to establish the trust based session.

[1] **https://www.simpleweb.org/wiki/index.php/Traces**.
[2] **http://www.caida.org/data/passive/ddos-20070804_dataset.xml**.

- All following traffic, correctly identified with verifiable tags, where the authenticity of the sender and the consent of the receiver are guaranteed.

If this was possible, then, the number of now possible attacks is limited to the possible attacks on the initial security message exchange protocol, and it's inherent variants. Moreover, it is also required to show that traffic not belonging to a safe message exchange would be detected.

The latter leads to a focus on attacks:

Based on legitimate looking initial exchange requests These attacks would be based on sending legitimate looking initial messages that don't intend to continue the session establishment.

Targeting the preliminary message exchange Attacks that would try to find exploits on an ongoing preliminary message exchange.

This analysis, necessarily of the white-box style, effectively limits the possible attacks spectrum, and at the same time allows for a more focused approach on the problem. In this chapter we present a quantitative effectiveness analysis of the mechanisms used by ELSP to establish its different variants.

In section 5.1 we discuss the design decisions of the previously presented protocol variants.

In section 5.2 we present a quantitative discussion on the effectiveness of ELSP in a reflection based scenario.

In section 5.3 we discuss ELSP's behaviour when faced with protocol exploit based attacks.

The chapter ends with section 6.5 where we conduct an overall assessment of the effectiveness of ELSP during the initial message exchange. We then discuss the conditions provided by ELSP to help in identifying with verifiable tags, that ongoing traffic belongs to an exchange correctly established.

5.1 ELSP Variants Analysis

In order to quantitatively predict the behaviour of ELSP in face of different DDoS attacks, we need to further define the parameters used in the message exchanges of the protocol, and also make concrete instantiations of the cryptographic mechanisms used.

In Table 5.1 we can see the cryptographic suites chosen for ELSP implementation. This will allow us to define the default length of all the messages present in ELSP. The reasoning for choosing each of the protocols depicted in the previous table will be further discussed in the following sections.

Also, to more accurately evaluate ELSP, we have approximated that 82 bytes (656 bits) will be needed for different protocol headers as show in Figure 5.1. The reasons for this choice will be addressed in the next chapter.

Table 5.1: Cryptographic suites selected for ELSP implementation.

Usage	Protocol
DH Key agreement	ECDH 256 bits
Keys and Signatures	RSA 1024 bits
Hash	SHA3 256 bits
HMAC	HMAC-SHA3 256 bits

Having set both a cryptographic choice and the size of the message headers, we are now ready for a quantitative discussion of ELSP in later sections of the chapter.

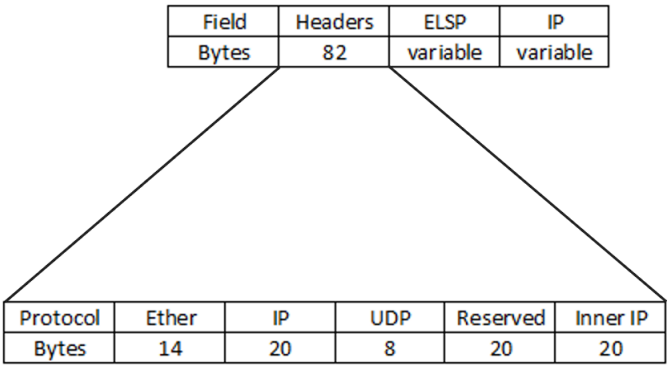

Figure 5.1: Headers decomposition for a ELSP packet.

In the following subsections we will present the exchange messages of the different variants in a more concrete manner, discussing the design decisions and offering a more detailed explanation than the one provided in Chapter 4.

5.1.1 Connectionless One Round

In this variant, the Initiator starts by randomly generating a "number only used once" (Nonce-I) with the purpose of being used as an exchange ID, i.e. if an entity knows all the current standing requests by the means of storing the Nonce-I value, it is possible to discard all incoming messages with an unknown nonce-I value. However, for the attack to be mitigated, the malicious traffic shouldn't even reach the victim, meaning that a dedicated upstream entity is available to filter all traffic and only forward legitimate traffic to the potential victim.

ELSP can be then be used to more easily mitigate reflection DDoS attacks by providing a filtering entity with the Nonce-I values and thus, forward only messages with the correct and current Nonce-I values to the potential victim.

As seen in Figure 5.2, the discrepancy in length for the I1 and R1 messages will fully depend on the connectionless protocol used. It's up to the Responder to decide if it's able

I1:

Field	Headers	Nonce-I	Connectionless request message
bits	656	256	variable

R1:

Field	Headers	Nonce-I	Connectionless reply message
bits	656	256	variable

Figure 5.2: Message composition for the Connectionless One Round Variant.

to immediately respond to the received query.

We then have two main mitigation mechanisms in a scenario used in this variant, the aid of a possible filtering entity upstream of the potential victim, with the use of Nonce-I and also, the RRL applied on the Responder side in order to mitigate reflection DDoS attacks.

5.1.2 Two Rounds with no Security Association

In this scenario, we have two fitting variants as depicted in Figure 4.1. As their differences in their default structure are minimal we will analyse them as one.

I1:

Field	Headers	Nonce-I	request message	Total
bits	656	256	request size	912 + request size

R1:

				Puzzle(I,K)		Echo Request		
Field	Headers	Nonce-I	PubKey-R	I	K	HMAC(Nonce-I,IP-I,I,K)	Sig	Total
bits	656	256	1024	256	8	256	1024	3480

I2:

				Solution(I,K,J)			Echo Request			
Field	Headers	Nonce-I	PubKey- I	I	K	J	HMAC(Nonce-I,IP-I,I,K)	request message	Sig	Total
bits	656	256	1024	256	8	256	256	request size	1024	3736 + request size

R2:

Field	Headers	Nonce-I	Reply message	Sig	Total
bits	656	256	Reply size	1024	1936 + reply size

Figure 5.3: Message composition for the Two Rounds with no Security Association variants.

The message equilibrium in this message exchange is based on the choices of the cryptographic suites. In order to both achieve cryptographic integrity and at the same time ease the burden on the Responder's side, low bit RSA were chosen.

In Figure 5.3 we can see a detailed composition of this message exchange. We can observe that the amplification factor between I1 and R1 will always be under 4 after factoring *request size*, this will be discussed in a later section.

The Responder when verifying the validity of a puzzle solution needs to check if the K $LSB_{H(I+J+IP-I+IP-R+Nonce-I)} = 0$, where LSB is the least significant bits, H denotes an hash function, "+", denotes concatenation and I is the random value given by the Responder, J the solution given by the Initiator, IP-I and IP-R the Initiator's and the Responders IP addresses respectively and the Nonce-I, the random number generated by the Initiator on I1 message creation.

A single computation of the previous hash, assuming a Skylake Core-i5 test platform at 2.7Ghz, takes 375 ns. Using probability theory we have, $P(1) = \frac{2^{256-k}}{2^{256}}$, as the probability of successfully completing the puzzle on one try. Thus, for average number of tries needed to complete the puzzle, we have 2^{k-1}. With an example value of k=16, 32768 tries would be needed on average to solve the puzzle, this implies a time of computation of $\approx$ 1,2 ms in the same test platform.

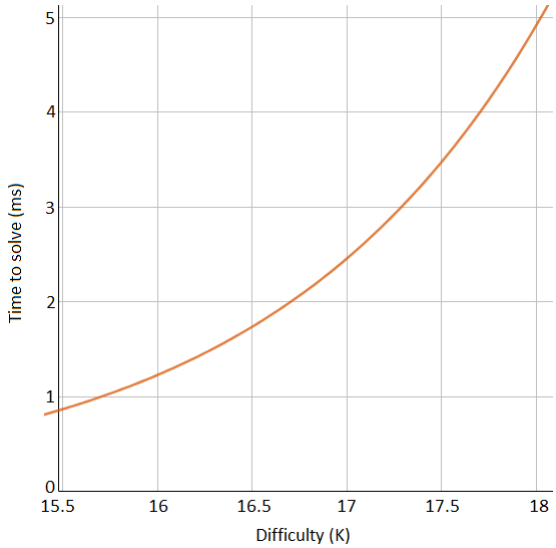

Figure 5.4: Time needed to solve the cryptographic puzzle as a function of its difficulty (K).

Before verifying the Initiator's solution, the Responder needs to first verify the Echo Request parameter. This parameter was previously HMAC'ed by the Responder and to verify it, the Responder creates another HMAC with a static private key only it has access to, and the parameters provided by the Initiator. If both HMAC's are equal, the puzzle solution is accepted and a reply message is generated.

5.1.3 Two Rounds with Security Association Variant

In this variant, it is introduced both the Diffie-Hellman algorithm and the exchange of Security Parameter Indexes. To ease the burden of the Responder, the usage of Elliptic curve Diffie-Hellman[10] was chosen in order to significantly reduce key size, a ECDH key of 256 bits has equivalent strength to a 3072 bit RSA key as referenced by NIST[3].

In this scenario, subsequent exchanged messages are identified by the session specific Nonce-I values, which are bound to security association parameters.

[3] **https://www.keylength.com/en/4/.**

I1:

Field	Headers	Nonce-I	Total
bits	656	256	912

R1:

				Puzzle(I,K)		Echo Request			
Field	Headers	Nonce-I	PubKey-R	I	K	HMAC(Nonce-I,IP-I,I,K)	DH-R	Sig	Total
bits	656	256	1024	256	8	256	256	1024	3736

I2:

				Solution(I,K,J)			Echo Request				
Field	Headers	Nonce-I	PubKey-I	I	K	J	HMAC(Nonce-I,IP-I,I,K)	DH-I	SPI-I	Sig	Total
bits	656	256	1024	256	8	256	256	256	32	1024	4024

R2:

Field	Headers	Nonce-I	SPI-R	$HMAC_{DH-KEY}$(Nonce-I, SPI-R)	Sig	Total
bits	656	256	32	256	1024	2224

Figure 5.5: Message composition for the Two Rounds with Security Association variant.

5.2 Effectiveness Analysis in a Reflection Attack scenario

As discussed earlier, with ELSP it is possible for a Responder to immediately answer a given request, and if the Responder decides to, it can instead send a cryptographic challenge to the Initiator to prove its good intentions. Thus, in a scenario where an attacker decides to use a ELSP enabled Responder, the traffic that reaches a potential victim is either a reply to the attacker given request, or a challenge that the potential victim did not request. With this in mind, and given the fact that high-amplification factor queries are automatically answered with a challenge, the maximum amplification factor can be derived from the inherent amplification factor within ELSP I1 and R1 messages.

In order to more easily model the total malicious traffic, let's introduce some notions:

Effective Attacker Requests (R_a) This can be obtained by multiplying the average bot upload rate and the total size of the botnet currently active.

Request Size(R_{size}) The size of the request message involved in the attack in bits, when not using ELSP.

A_f The amplification factor.

R The number of reflectors the bots are using to amplify traffic.

RRL Shaper(R_{shap}) The number of packets per second (pps) allowed by the RRL mechanism.

Puzzle Message Size(P_{size}) The size of the challenge message given by the Responder.

Also, for a clearer analysis, we shall establish three different scenarios:

1. **A baseline scenario with no shaper or ELSP usage.**

   A DDoS reflection attack scenario with no shapers can be approximated by:

$$R_v = R_a * A_f$$

This scenario is devastating when we take into account the possible amplification factors for protocols such as DNS and NTP that can be in the tens to few hundreds range, with higher values being more difficult to attain. However, this scenario is heavily limited by the ability of the reflectors to respond such vast a vast and unrestricted amount of queries.

2. **Using a shaper policy and no ELSP usage.**

   When a shaper is in place on all used reflectors, the attack can be modelled as:

$$R_v = R * R_{shap} * R_s * A_f$$

   However, this model is only valid when R_a is enough to maximise the shaper throughput in all reflectors, as in:

$$\frac{R_a}{R} > R_{shap} * R_{size}$$

   The strength of the attack is now based on number of reflectors, the amplification factor and the shaper policy, but not by the attacker's upload power. This is because the shaper mechanism only allows a fixed amount of outgoing traffic.

   However, when faced in a situation without ELSP , the traffic received by the victim (R_v) depends on many other parameters. We now need to take into account the total upload power of the attacker as well as the amplification factors inherent to the connectionless protocols used

3. **Using ELSP.**

When using ELSP if a Responder successfully detects a reflection attack, the following traffic to the victim can be modelled by:

$$R_v = R * R_{shap} * P_{size}$$

We can disregard R_a and A_f because, the outgoing traffic of a Responder with RRL only depends on how many packets per second he is willing to send to a certain IP and the size of this packet, in this case it's the Puzzle Message Size.

Thus, the main difference between scenario 2 and scenario 3 where ELSP is used, is in the maximum amplification factor allowed, in the case of ELSP, as seen before this value is always under 5.

This ceiling value for the possible amplification factor heavily limits the usage of ELSP as a protocol for reflection based DDoS attacks. When comparing with the values discussed in Table 5.2 we can see a weighted average amplification factor value of 124, which is ≈25 times worse than using ELSP, thus reaching the conclusion that if an attacker wants to perform a reflection based DDoS attack it wouldn't use ELSP Responders for it, and as such, ELSP enabled entities wouldn't be used as reflectors.

Table 5.2: Maximum known amplification factors for the most common protocols used in reflection DDoS attacks.

Protocol	Max. known Amplification factor	Share in Reflection DDoS(%)
DNS	70	40
CLDAP	57	21
NTP	206	19
CHARGEN	360	12
SSDP	30	8

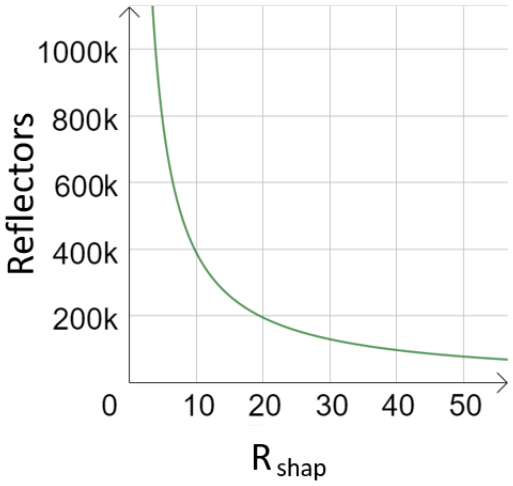

Figure 5.6: Number of reflectors needed to achieve a typical DDoS reflection attack of 12Gbps using ELSP, as a function of R_{shap}.

5.2.1 Attack bandwidth example instantiation

For a more concrete idea of the impact of the previously discussed scenarios, let us compare computed attack traffic, in all three scenarios, generated by a 1Gbps of botnet upload capacity, using 1000 reflectors with each one accepting at most 10 requests per second, and having a request size of 784 bits (98 bytes), value obtained in the same fashion as the amplification factors obtained in Table 5.2, for the same protocols.

We then have:

Table 5.3: Resulting Attack bandwidth, R_v, in an example instantiation where we have 1Gbps of initial botnet upload capacity, 1000 reflectors each one accepting 10 packets per second and having a request size of 784 bits.

Scenario	Formula	R_v	$A_f = 124$
1	$R_a * A_f$	1 Gbps * A_f	128 Gbps
2	$R * R_{shap} * R_{size} * A_f$	7,84 Mbps * A_f	0,97 Gbps
3	$R * R_{shap} * P_{size}$	33,6 Mbps	33,6 Mbps

Table 5.3 further demonstrates the importance of a shaper mechanism.

5.3 Effectiveness Analysis in Protocol Exploit based Attacks

When using ELSP, the viability of most protocol exploit based attacks is severely reduced.

SYN Flood This attack relies on exhausting the table slots available for starting TCP connections by flooding the victim with spoofed SYN messages. With ELSP, for an Initiator to start a TCP connection it first need to complete a cryptographic puzzle, thus fully mitigating this attack vector. As the R1 messages on the Two round with security association variant are pre-signed to avoid on the fly Diffie-Hellman and RSA computations, one can argue that an attack of similar type can be exploited within ELSP, this is, spoofing I1 messages to try and exhaust the available pre-signed R1 messages.

UDP Fragmentation As ELSP is a network layer DDoS mitigation protocol, it's possible to enforce the usage of lower than MTU packet sizes and thus fully mitigating this attack vector. In fact, in all variants except the two rounds with no security association variants, all messages in ELSP exchange are smaller than 3400 bits (425 bytes).

As the minimum datagram size required to request fragmentation is 576 bytes[16] and, messages in ELSP are all lower than this value, it is possible to correctly discard all ELSP messages with fragmentations tags, as these would surely be malicious.

5.4 Overall effectiveness and conclusion

In the beginning of this chapter, we stated that ELSP would be effective in mitigating reflection, protocol exploit and flood based DDoS attacks if a correct effectiveness analysis of both the traffic needed to establish the protocol and all following traffic, where the authenticity of the sender and the consent of the receiver, are guaranteed.

Let's begin with the first statement, the effectiveness of the base exchange of ELSP is correlated with the absence of a viable attack vector through it. And, as discussed on Chapter 4 and on Sections 5.2 and 5.3 no new vulnerability is introduced with the usage of ELSP. More explicitly, we discussed a number of possible attacks on the protocol, and the resilience of ELSP to these attacks would determine its effectiveness. These attacks and the behaviour of ELSP in their response are:

- **Based on legitimate looking initial exchange requests** These attacks are based on sending requests that don't intend to continue the protocol. They try to exhaust the resources of the Responder in order for legitimate Initiators to not be able to communicate with them. In ELSP, this attack would be best used on the two round with security association variant, where R1 messages are pre-signed, and the goal

of the attack would be to exhaust this pre-signed message. The ELSP solution is similar to the one employed by HIP, when in a situation of exhaustion of pre-signed messages, the Responder uses the available signed Diffie-Hellman values to more than one Initiator.

- **Targeting an ongoing message exchange** In this category of attacks we have the reflection attacks discussed in Section 5.2, and also attacks such as the I2 flood and R1 replay discussed in Chapter 4.

For the second statement, we need to discuss if the traffic following the base message exchange produces vulnerabilities. This would only make sense in the variants two round with security associations and a connection oriented two round with no security associations. For the first variant, the usage of a well established protocol like IPSec dismisses all vulnerabilities within the proposed Attackers model discussed in Chapter 4. For the second variant, we have a scenario where, an Initiator wants to connect to a Responder via TCP, to accomplish this, first the Initiator needs to complete the protocols base message exchange, which includes completing a cryptographic puzzle, if in this scenario the Initiator has malicious intent, he could try to attack via e.g. SYN flood. In this example however, the effort used by the attacker is much greater than the damage caused (completing a demanding cryptographic puzzle vs exhausting one table entry).

Another by-product of ELSP is its positive impact on the functioning of a firewall. Without ELSP usage, a firewall has the strenuous task of checking a large set of rules to each incoming packet, with ELSP it's now possible for the firewall to only check if the message is a ELSP message and if the Nonce-I is known. These two verifications are enough to achieve comparable firewall effectiveness levels of much more resource intensive filtering techniques.

Chapter 6

Implementation proposal and Proof of concept

Having analysed the functionality, mechanisms and the effectiveness of the ELSP variants in the previous chapters, in this chapter, we shall now discuss possible implementation and adoption barriers as well as introduce a guideline for ELSP implementation.

The first barrier we found is the NAT traversal problem, also present in HIP. In Section 6.1 we discuss the problem on HIP especially as well as present its current solution. Then, in Section 6.2 we discuss a general purpose approach of this previously discussed solution and argue other possibilities.

The second implementation barrier we found is in the interaction of ELSP with applications and is discussed in Section 6.3 alongside the guidelines for ELSP implementation.

Finally in the last section of this chapter we will discuss the key points of the proof of concept implementation provided in I.

6.1 Problems regarding NAT traversal

In the current Internet, edge, simple pure IP end-to-end paths are rare. For several reasons, devices modify or extend the forward functionality of the Internet. These are often called *middleboxes*[1], and they affect Internet communications in many ways, e.g. they inspect protocol flows and drop, insert or modify packets.

There are many different types of middleboxes, such as network address translators (NAT) and firewalls. Hosts behind NAT-enabled routers do not have end-to-end connectivity and as such cannot participate in some Internet protocols, e.g. services that require TCP connections from outside the NAT-ed network.

[1] Any intermediary device performing functions other than the normal, standard functions of an IP router on the datagram path between a source host and a destination host[2]

Because when using a NAT-enabled router, all hosts behind it are all seen publicly as a single public IP address. Only traffic directed to a non NAT-ed network does not encounter NAT traversal problems.

6.1.1 Case study: HIP

HIP suffers with two main problems caused by middleboxes. They interfere with the base exchange and with the transmission of HIP data traffic carried within the Encapsulating Security Payload (ESP). The protocol specification for HIP requires an additional header at the network layer, which is necessarily delegated to the *next protocol* IP header. If a protocol number is not recognized, the packet is rejected.

An extra complexity may arise with NAT, since NAT, in general may also interfere with the transport header.

HIP's traverse success depends on the type of NAT used, if the NAT only interprets and modifies the IP header and does not inspect the IP payload, the array traverse occur without problems. However, these basic NATs are rare[37], much more common are the NATs that inspect and translate transport-layer port numbers, and because the IP payload used in HIP BEX does not contain port numbers, these NATs cannot relay on them.

In HIP, after BEX is successfully completed, subsequent data exchanges between the two parties will use ESP. Thus, HIP faces the same challenges that IPSec faces with regards to NAT traversal.

ESP-Encrypted data traffic makes it so that all upper layer headers are invisible to a NAT, and thus, changes to the IP header will invalidate upper layer checksums that cannot be changed, as they are within the ESP-protected payload.

As such, ELSP implemented directly at the IP network would face the same challenges as HIP regarding NAT traversal.

Currently as a work in progress document, [13] discusses a possible solution. In a HIP architecture, for two hosts to communicate over NAT environments, they need a reliable way to exchange information, for this purpose and to cover the general case, i.e. both hosts are behind NATs, an "HIP Relay Server" was defined. It works by supporting the relay of HIP control messages over UDP via tunnels. Figure 6.1 further illustrates this solution.

6.2 Tunnelling over UDP as a possible solution and alternatives

A tunnelling protocol is based on the principle of encapsulation. It allows a foreign protocol to run over a network that does not support such protocol, and works by using the data portion of a packet to carry the unsupported protocol.

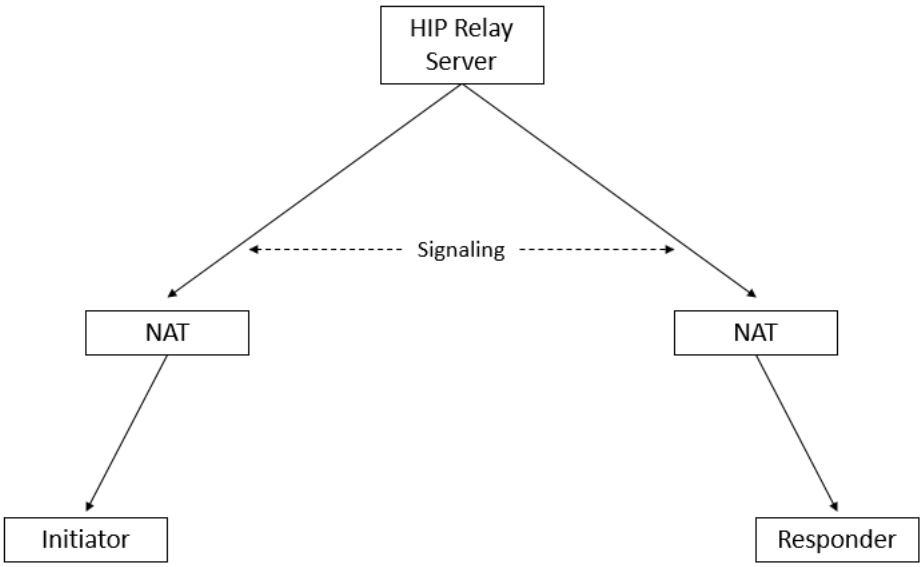

Figure 6.1: Example network configuration, where both the Initiator and Responder of a HIP exchange are behind a NAT.

As such, it's possible to e.g. send MAC frames or IP packets to different networks connected over the Internet, by having the entire frame or packet stored in a UDP data payload. Further examples will be discussed in the following sections.

As we have seen, building ELSP by simply having a ELSP header as a *Next Header* protocol in a IP packet is not realistic. Furthermore, such a scenario would require IANA[2] approval.

Moreover, ELSP could only be used where it would be generally adopted, as both hosts need to support ELSP, thus, it would face the same NAT traversal problem HIP faces. As such, a much more easily approachable and still valid solution would be to encapsulate ELSP messages via a UDP tunnelling mechanism.

Tunnelling can then also possibly be a solution for the previously discussed NAT traversal problem, by having ELSP headers encapsulated within a commonly used transport layer protocol that doesn't have problems with NAT traversal, it's then possible to make transparent usage of such protocol.

However, due to the nature of the demultiplexing present in NAT devices, the former is only valid when only a single host is behind the NAT. This occurs because, for the encapsulation and de-encapsulation that a transport layer tunnelling protocol requires, a single port number needs to be reserved and thus only one host behind each NAT.

This is not acceptable, as there are commonly multiple hosts behind each NAT, as such, using tunnelling for ELSP implementation is only useful for bypassing firewalls

[2]Internet Assigned Numbers Authority.

and NATs in the sense that ELSP as *Next Header* protocol would be immediately discarded as it wouldn't follow common firewall rules nor, being in the network layer, have a port number and thus would be discarded also by the NAT device.

The current solutions for NAT traversal are based on the same defining principle that a relay server is needed for efficient communication between two hosts that are behind NAT devices. Examples of protocols implementing this principle are:

Socket Secure(SOCKS) SOCKS is a protocol that exchanges packets between client and server using a relay proxy, additionally it also provides security features for accessing the proxy. Practically, with SOCKS, UDP packets and TCP connections are forwarded via a proxy server.

Traversal Using Relays around NAT (TURN) TURN [17], unlike SOCKS, is a general purpose protocol specialized in traversing NAT devices. A complete, general purpose NAT traversal solution, requires a means by which a client can obtain the address from which it can receive data from any peer in the public Internet. This requires a server available on the public Internet to relay such information and TURN is the protocol that allows a client to obtain the required IP addresses and ports from such relay server.

In a real world scenario, such NAT relay servers should normally be found along the node paths of the two communicating peers, as such, extra communication hops would not necessary.

In summary, even if currently, the implementation of ELSP at the network layer as a *Next Header* IP protocol were realistic, it would still require a *middlebox* traversal solution. Tunnelling protocols do not, per se, solve NAT traversal problems, but are effective in solving problems resulted from other types of *middleboxes* such as firewalls.

Thus, it seems that any realistic ELSP implementation needs a tunnelling-based solution. We shall now review some of the most appropriate tunnelling protocols for conjoined usage with ELSP.

6.2.1 LISP

The Locator/Identifier Separation protocol[4] was designed with the purpose of solving the problem of suboptimal route scaling present in the single IP address paradigm, and also the problem of multi-homing.

Its proposed solution is based on an architecture where a host has two distinct IP addresses: one to indicate routing locators (RLOC) for traffic through the global internet, and a second one for endpoint identifiers (EID), used to identify network sessions between devices. LISP relies on tunnelling based on a UDP encapsulation solution.

Figure 6.2 displays a general overview illustration of a LISP deployment environment, showing the three essential LISP environments: LISP sites (EID namespace), non-LISP sites (RLOC namespace), and the LISP mapping service (third-party infrastructure).

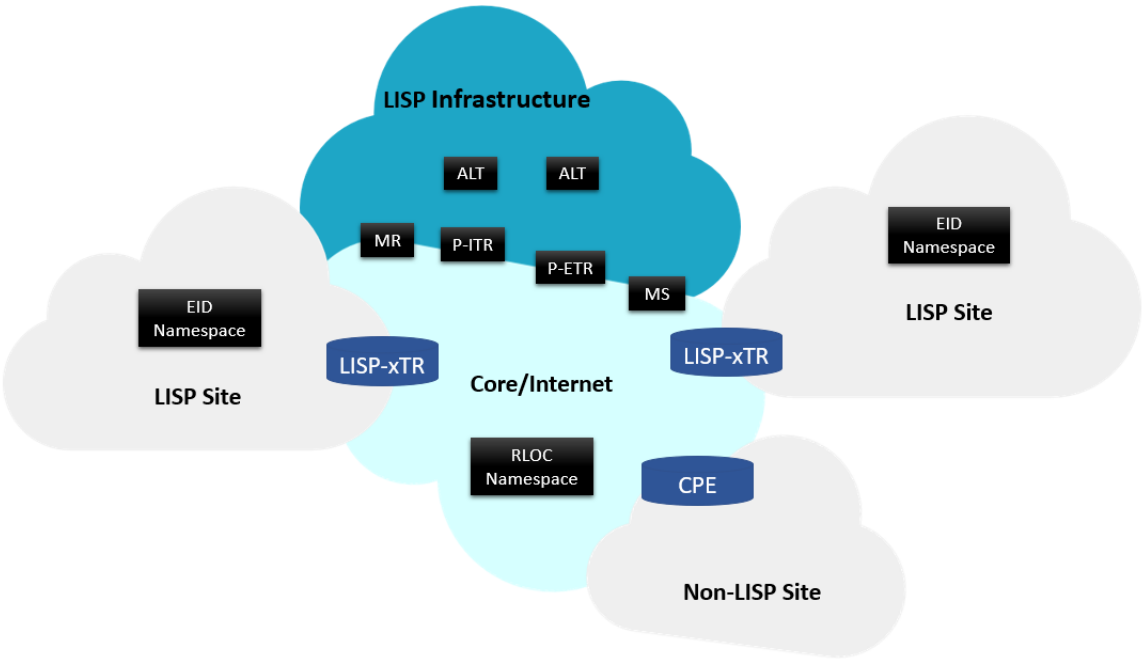

Figure 6.2: Overview of a LISP deployment.

LISP EID namespace represents the architecture's host space, these are connected to the global Internet by endpoint routers (LISP-xTR). IP addresses used within these LISP sites are not advertised to the non-LISP Internet.

To fully implement LISP, several additional infrastructure components are needed as part of the deployment. These devices function in the LISP roles of map resolver (MR), map server (MS), proxy egress tunnel router (PETR), proxy ingress tunnel router (PITR), and LISP alternative logical topology (ALT).

Although LISP is fully standardized and implemented in Linux and Cisco routers, it's not a popular protocol and lacks wide spread adoption.

Also, LISP, as a protocol within the Locator/Identifier separation paradigm is not the desired choice in this scenario, as end-to-end connectivity is one of the top priorities for ELSP.

6.2.2 GRE over UDP

This protocol[41] allows for the encapsulation of IP packets within GRE and UDP headers. The Generic Routing Encapsulation is a simple IP over IP tunnelling mechanism developed by Cisco and later standardized[8].

Encapsulation and De-Encapsulation occurs on the network switch. When a switch receives a IP packet to be tunnelled, it sends the packet to the tunnel interface, this tunnel interface then encapsulates the packet in a GRE packet and adds an outer IP header and a UDP header in case of GRE over UDP. When a switch receives a message from a tunnel

interface, the outer IP header, the GRE header and the UDP header are removed and the packet is routed based on its inner IP header.

GRE over UDP is then an extension on the widely established GRE protocol that allows traffic to go through *middleboxes* such as NATs.

6.2.3 Virtual Extensible LAN

Data centers are often required to host multiple tenants, each with their own isolated network. As such, VXLAN emerged from the need to increase the maximum number of hosts in Virtual LAN (VLAN).

VXLAN is a tunnelling protocol that encapsulates Ethernet Layer 2 network services in UDP datagrams, where the original L2 frame has a VXLAN header added and is then placed in a UDP packet. With this MAC-in-UDP encapsulation, VXLAN can tunnel Layer 2 network over Layer 3 networks, effectively allowing normally intra-network traffic across different networks and increasing the maximum number of hosts in a VLAN from 4094 to over 16 million.

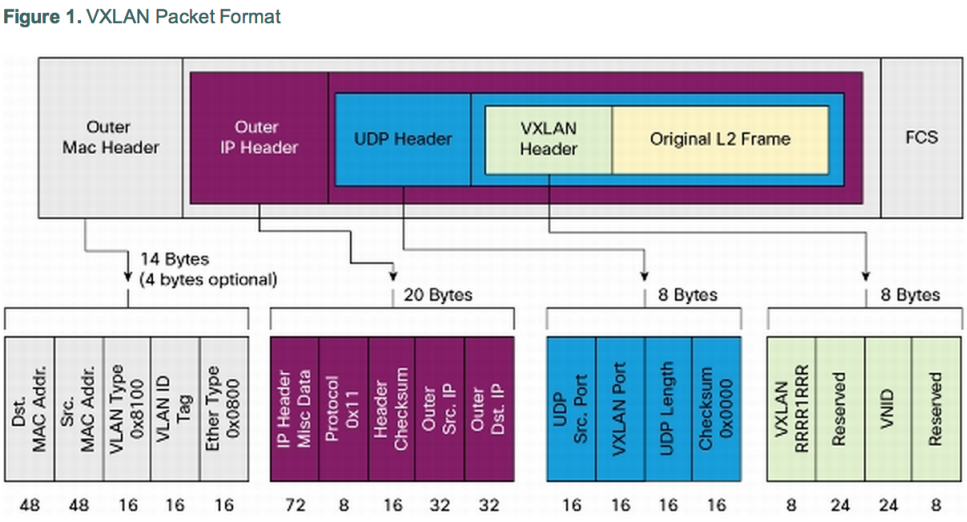

Figure 6.3: VXLAN Packet format.

In a VXLAN architecture, each VXLAN overlay scheme is termed a VXLAN segment, and only hosts in the same segment can communicate with each other. Each VXLAN segment is identified by a 24-bit VXLAN Network Identifier (VNI) which provides network isolation.

Each Ethernet frame is encapsulated according to a set of rules in the VXLAN packet format as seen in Figure 6.3, encapsulation and de-encapsulation occurs at the Tunnel Endpoint (VTEP). VXLAN related tunnel and header encapsulation information are only known to the VTEP. Thus, usage of VXLAN, as seen in the host's perspective is transparent.

6.2.4 How Tunnels are integrated in Operating Systems

TUN and TAP are virtual network kernel interfaces, supported entirely in software and provided by all Unix derived kernel operating systems (Linux, Mac OSX, BSD), and also supported in windows by the OpenVPN project. TUN simulates and operates with layer 3 packets, while TAP simulates a layer 2 link layer i.e. Ethernet frames.

Packets sent by the operating system to a TUN/TAP interface are delivered to a user-space process which attaches itself to the interface. User-space programs can also pass packets into the TUN/TAP interface they are connected to, effectively injecting them into the operating system's network stack.[3] In this case, the operating system believes it received the packets from an external source, thus making possible a transparent usage by applications.

Virtual Private Networks e.g. OpenVPN, OpenSSH, and virtual machine networking such as Open vSwitch rely on TUN/TAP implementations. As such, there are TUN/TAP or equivalent drivers for all the major operating systems, its portability however is operating system specific.

6.2.5 Choosing an adequate approach

Mostly, any standardized tunnelling protocol that would resist passing through the middleboxes described earlier, would be a possible candidate.

Thus, we give high importance how generalized the adoption is for a given tunnelling protocol.

As seen earlier LISP, even though it has interesting features, would not be a good fit to be used with ELSP.

GRE over UDP offers a simple, and achievable approach to an encapsulation mechanism. GRE over UDP, also mentions in [41] considerations to help solve its middlebox related problems. However, its implementation is not yet standard nor it is widely implemented in all operating systems.

Even though VXLAN is aimed at MAC over UDP encapsulation, its usage with ELSP is suitable. The general implementation and deployment of VXLAN over the Internet using open source software such as Open vSwitch (ovs) satisfies ELSP goal of friction-free protocol adoption. Other favourable VXLAN characteristics are its dominant usage on the Cloud as well as its implementation in all current operating systems. One setback we should point about VXLAN is its higher overhead compared to protocols like GRE over UDP (50 bytes instead of 28 bytes in GRE over UDP).

The chosen solution in this section is agnostic to the discussion present in the next section. However, we believe that VXLAN would currently be the best decision to be used with ELSP, and calculations made in the previous chapters took this into account.

[3] This is accomplished transparently when the routing table of the host sends IP packets with the appropriate destinations through that interface

Such solution would consist of a fixed port VXLAN tunnel in which NAT traversal would be aided by general purpose, public NAT relay servers found along the communications path.

6.3 Integrating ELSP with applications

For a transparent integration between applications and ELSP, the interactions made by the applications need to be oblivious of the presence of ELSP. Otherwise, the applications would need to be modified, which is not a realistic approach.

For this, information going and outgoing the application needs to be dealt with by an intermediary in the host. For this purpose, a suitable and easily integrated solution for most operating systems would be to rely on the usage of an ELSP daemon at user-level, responsible of making logical decisions on how to behave in different scenarios. Such daemon, would work in a way that it would intercept and re-route packets sent by applications to the daemon via routing rules. On the other end, as communications are based on a tunnelling protocol, incoming traffic is easily distinguishable and also routed to the ELSP daemon. This is illustrated in Figure 6.4

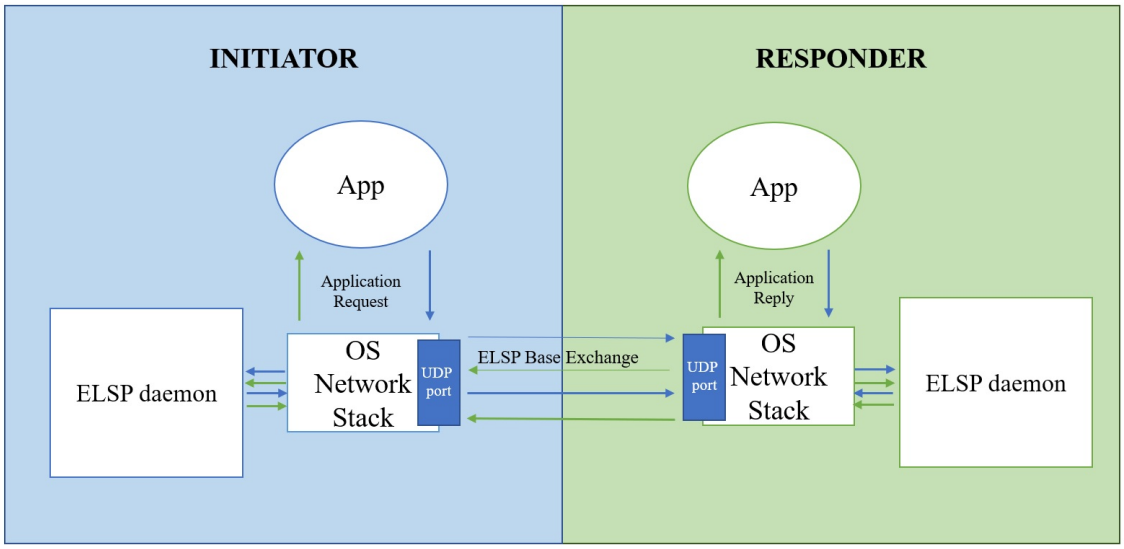

Figure 6.4: ELSP tunnelling process implementation overview.

The ELSP daemon also handles the protocols inherent logical state machine consisting of the base exchange, managing previously known ELSP enabled or disabled addresses, and all other necessary processing needed for having the application act transparently as if it were communicating directly to the operating system network stack.

Only the hosts IP packet routing needs to be modified by its administrator. In a first approach, we can assume that all incoming traffic is sent to the ELSP daemon.

All outgoing traffic is sent to the ELSP daemon and depending on it, it is sent via a tunnel or directly to the network stack for normal routing. All ELSP related traffic is

directly sent to the ELSP daemon to be process and, when appropriate, de-encapsulated and routed to the local application.

If the ELSP daemon receives a packet from the local host (i.e. it has been routed from the application to the tun/tap interface used in the tunnelling protocol), it is an application packet and therefore the daemon must realize if the packet is the first in that applications recent exchanges or it belongs to a known ELSP association.

If the received packet is a first of a given association, the daemon must decide if it will use ELSP itself or simply send the request traditionally. This settings allows for flexibility. To complement such flexibility, ELSP should have a probing mechanism that records different external hosts behaviours when dealing with ELSP. On the contrary, if the packet is from a known association, the daemon will act on it based on the previously discussed logical behaviours on Chapters 4 and 5 (i.e. deciding which protocol variant to use as well as decide the correct response to the given protocol step).

This probing mechanism first detects if the given target host has its ELSP ports open. If closed, the host is immediately regarded as non ELSP compliant for a set amount of time. However, if the ELSP ports are open this still does not mean that the host is ELSP compliant.

If a ELSP I1 message is sent and no response is received within the timeout time frame, the host is also added to the known non ELSP compliant hosts for a certain amount of time. When an application tries to communicate with known non ELSP compliant hosts, the daemon treats the request traditionally.

If the packet reaches the ELSP daemon via the network, the daemon must then also decide if the packet belongs to a new or to a known ELSP association. ELSP associations are identified by IP addresses and Nounces and also have corresponding timeout values, that if exceeded delete the association from the current associations table.

Also to be considered at this stage, is whether the received packet is in fact ELSP compliant, if it's not compliant for reasons such as header malformation or wrong message type in junction with the current state in the ELSP state machine, the packet is simply discarded. On the other hand, if these parameters are in conformance and correct, the daemon reacts accordingly to the current stage of the ELSP BEX or the already established ELSP association.

For a simpler explanation of the rules used, we will separate them into different scenarios: UDP traffic and TCP traffic. Transport layer protocols other than UDP and TCP will not be considered at this time.

6.3.1 ELSP daemon execution of ELSP UDP BEX

When receiving a first I1 message or when trying to initiate an association with an I1 message, the ELSP daemon inspects the transport layer protocol and in conjunction with other administrator chosen security settings, the daemon chooses the ELSP variant to

continue. Figure 6.5 shows the the ELSP guidelines for when the transport layer protocol is UDP.

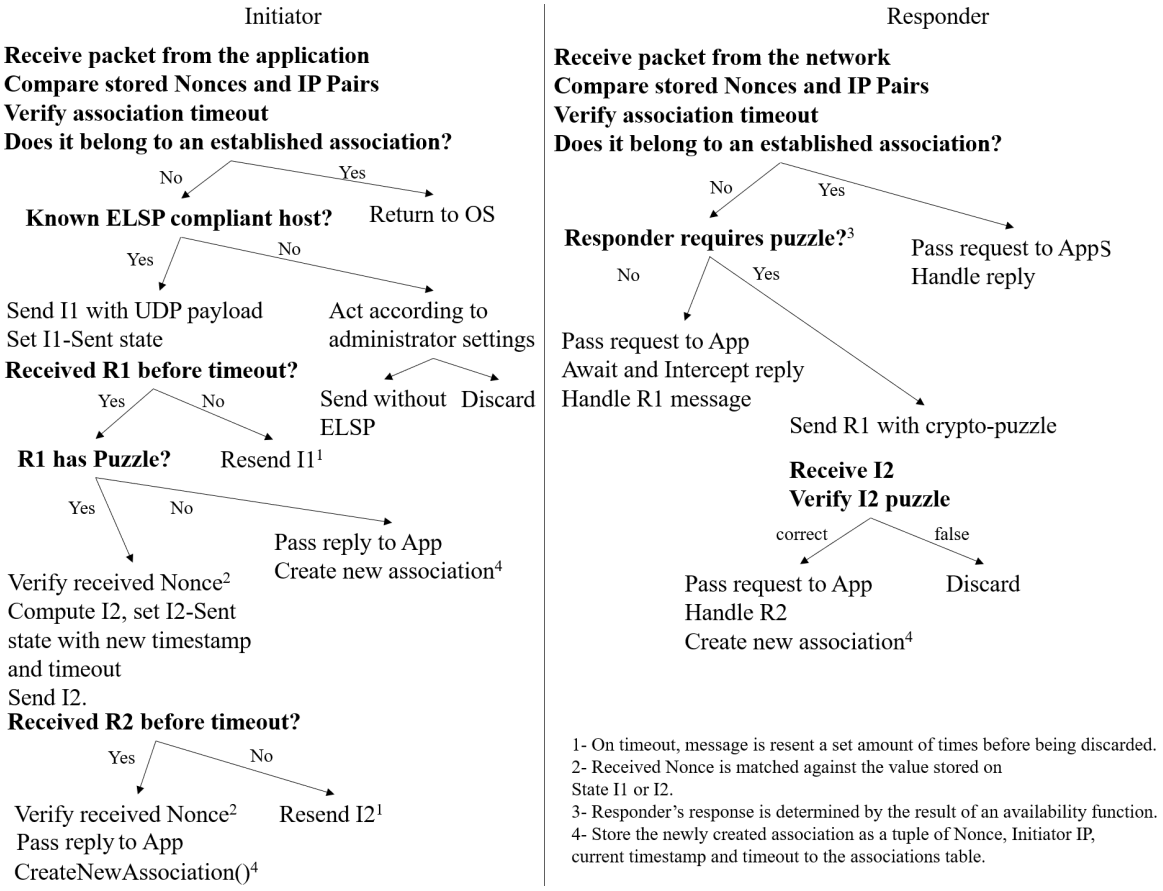

Figure 6.5: ELSP UDP logic flow overview when a new association is locally initiated (left) and when it is remotely treated (right).

The number of times a message is re-sent after having its timeout triggered and the amount of time to wait before re-trying should enable the occurrence of lost packets in the network without damaging the functioning of the ELSP protocol.

The Responders choice on whether or not to challenge the Initiator with a cryptographic puzzle should be made based on factors such as current load or whether or not it's RRL flagged for usage as a possible DDoS reflector. These metrics should be made accessible by the ELSP daemon, either directly or via a specialized API.

6.3.2 ELSP daemon execution of ELSP TCP BEX

If the inspected transport layer protocol is TCP, then the ELSP daemon needs to react in a different way, always choosing a two round variant, with or without security associations depending on externally established parameters (by an administrator or any other mean).

Figure 6.6 shows the rules implemented when dealing with a TCP connection. However, TCP is a complex protocol and shown is only a simplified instance of a real world scenario.

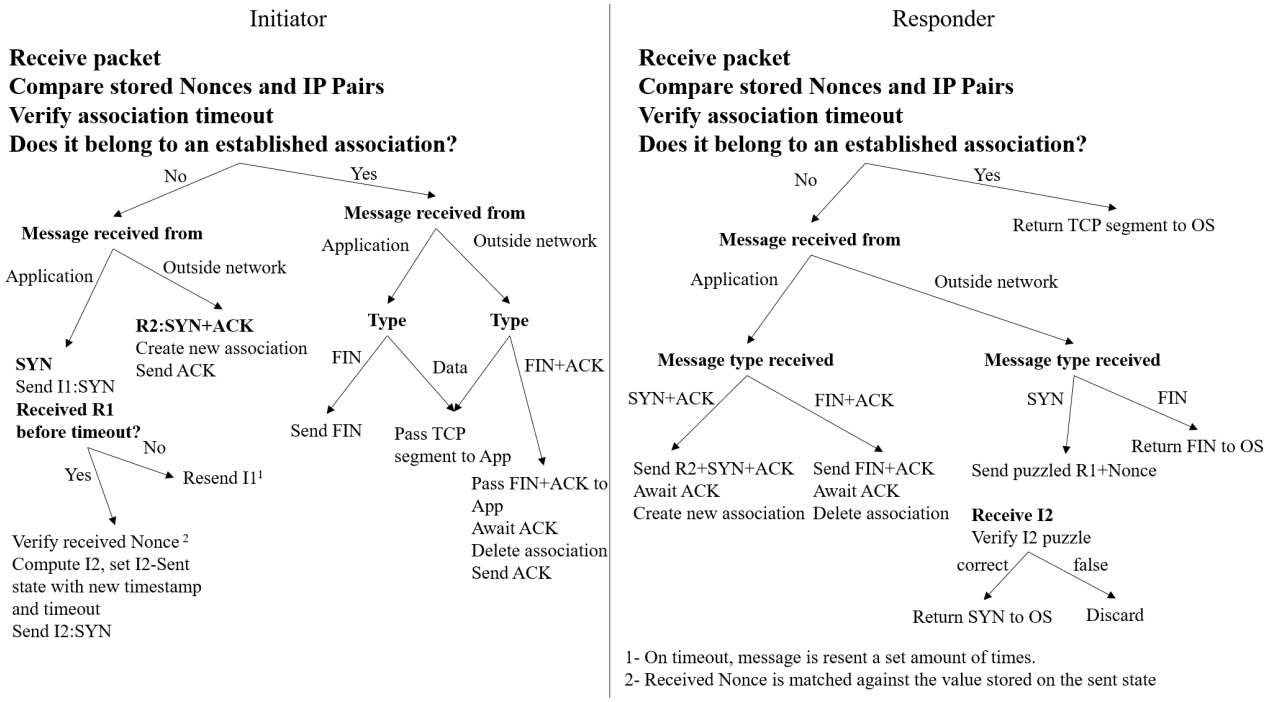

Figure 6.6: ELSP TCP logic flow in case of success.

In a simplified manner, the ELSP tunnelling process intercepts and interprets the usage of the TCP segment before being sent on the Initiators side and when being received on the Responders side.

Before a connection is deemed established in a ELSP setting, the ELSP base exchange first needs to complete, as such, a TCP SYN segment triggers the R1 message, the corresponding TCP ACK is returned only after the puzzle message is confirmed on the R2 message, and as TCP is a 3 way handshake, the Initiator will send a SYN+ACK to symbolize connection establishment.

On the Initiators side, after the ELSP daemon intercepts a packet and identifies it using the TCP protocol, it then compares the received packet to the table of current associations via Nonce and IP pairs and also checks for a timeout. Then the ELSP daemon takes into account from where was the packet received from, which is an early indication of the possible TCP message types.

On the Responders side, the process is similar, taking most importantly into account the state transitions imposed by TCP. As seen in 6.7, to reduce on overhead, SYN and SYN+ACK messages are “piggybacked” on the ELSP BEX.

6.4 ELSP Proof of concept Implementation Overview

In I we can find our proof of concept implementation of the ELSP daemon written in python, for the UDP protocol case. It presents outgoing packet capturing, successful ELSP base exchange and original packet delivery to its destination. It also supports

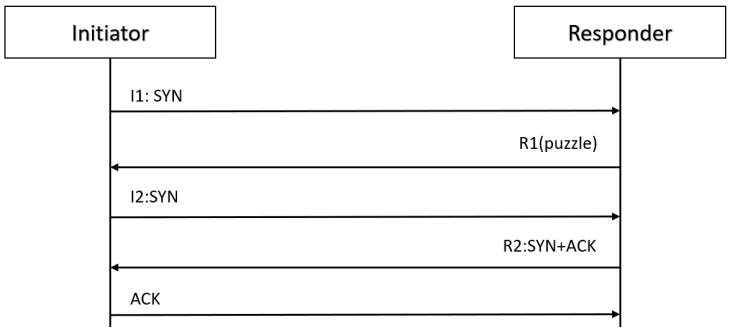

Figure 6.7: Temporal flow of a successful TCP connection with ELSP.

multiple associations per host. This proof of concept represents the two round no security association variant, triggered on connectionless outgoing datagrams (UDP).

Let us now overview the key components in our ELSP daemon proof of concept.

Libnetfilter-queue is a library that provides user-level access to packets filtered by the kernel packet filter. This library comes as standard for many of the most popular Linux distributions. NetfilterQueue is the python library that provides the python bindings for Libnetfilter-queue.

Iptables is a command line utility for implementing Linux kernel firewall rules. With Iptables and Libnetfilter-queue, it's possible to add incoming and outgoing firewall rules to re-route packets into a numbered queue, user-level accessible.

The path a packet follows when an outgoing packet is captured by the ELSP daemon is then as follows. Having set both incoming and outgoing iptables rules (either for a certain IP range, or for all incoming and outgoing addresses), a packet is then routed to the NFQUEUE numbered on the iptables rule. On the ELSP daemon, when a packet arrives on the queue, a callback function is called that filters the packet as either incoming or outgoing, and thus the ELSP base exchange occurs. When a packet arrives in the queue, it is treated in the ELSP daemon until it is either accepted or dropped. While the ELSP BEX doesn't complete, needed informations are stored on the pendingAssociations table, that is responsible for storing the current ELSP state and on the Initiator's side it also stores the Initial requests and the received Responders RSA public key materials.

On the last stage of the ELSP BEX, the I2 message carries the original request captured in the Initiators host, which when the Responder deems this I2 message authentic and correct, the Responder removes the ELSP related fields on the received IP packet and only leaves the original request and then accepts this modified packet to be delivered to the corresponding application.

After accepting this recently modified packet, the responder can now add a new entry on the associations table with the peer's IP address and corresponding message Nonce value. All the packets captured by the ELSP daemon that have its target IP address in the associations table is immediately accepted.

6.5 Conclusion

The definition and implementation of a network level protocol is a challenging task, having many, hard to predict, barriers to look out for. ELSP is no different and in this chapter we discussed a few of the implementation barriers to be expected with its implementation.

In this chapter we have discussed ELSP implementation guidelines and the way it overcomes its adoption barriers. We also presented practical solutions for these problems and choose to delve into a solution based on a custom ELSP tunnelling process that creates a tunnel between two hosts, facilitating ELSP message flows.

We have shown the logical flows for both UDP and TCP communications. However, we have not discussed communications based on the Two round with Security Association variant as the differences are based on the ELSP protocol logic and as such was discussed in the previous chapters. By default, the ELSP variant with security associations is disabled, and its usage can only be enabled by the hosts administrator. Both hosts in a Security Association based ELSP connection need to have this setting enabled.

Chapter 7

Conclusion

7.1 Conclusion

Nowadays, DDoS attacks represent a significant amount of all attacks that take place in the Internet, which leads to significant economic losses, especially when compared to the much lower required investments by the attackers.

In this dissertation we have delved into the ever growing problem that is DDoS in the current state of the Internet. We presented an overview of the state of the art detailing the different exploits attackers use to inflict loss of availability on its victims.

Also, as a counterpart, we discussed the current efforts in DDoS mitigation technology, and how they are not enough to combat such an ever evolving problem. Among these, the most popular are based on server replication and hiring specialized cloud-based absorption companies to load balance incoming attacks. Both these choices are bound to be disproportionate in what concerns cost of defence versus the cost of the attack, as new exploits keep surfacing with increasingly more effective results, and need less and less resources from the attackers while fortunes are spent by businesses.

Radical changes need to happen if we want DDoS to be a thing of the past. However, current least effort ideology practises taken by ISP's mean that all and any changes proposed are all taken with great friction.

The current solution based on shielding critical DDoS attack targets behind cloud mega-infrastructures does not solve the crux of the problem, but instead only patches it, also contributing to a decreasingly end-to-end Internet.

We presented the Host Identity Protocol as a case study, even though it contains several very effective security and DDoS mitigations mechanisms, it still hasn't received wide adoption. The reason for this, is the fact that HIP was designed to try to solve several orthogonal problems at the same time, e.g. it both tried to solve security, authentication

problems as well as multi-homing and universal connectivity. This is a too complex requirements list if taken all together.

As such, in this dissertation we propose ELSP, a network layer protocol inspired from HIP and designed to address the DDoS issue in an easy to adopt approach without losing security functionalities and mitigation effectiveness. We studied the factors that are taken into account to determine the severity of the influx traffic received by the victim, and provided an analysis based on the theoretical models.

With our analysis we concluded that in a reflection based DDoS attack, using ELSP and RRL was up to 25 times more effective than current situations where only RRL mechanisms are used. For the other main DDoS attack vectors discussed in this document, we have shown that when in usage of ELSP, both UDP Fragmentation and SYN Flood would be mitigated.

We predict that the DDoS problematic will continue to grow, as the current solution of mostly increasing resources, is not a sustainable solution. As such, it's up to the scientific community to research and test the effectiveness of less demanding solutions.

We hope that the work presented in this document helps to prove that non intrusive and effective solutions for this important and omnipresent issue are possible.

7.2 Future work

In this section we present possible future work regarding the protocol present in this document.

Complete implementation of the ELSP logic core In this work, we present a working proof of concept, that indeed proves possible DDoS mitigation. However, a production ready version of ELSP will still need support for more protocols such as TCP and ICMP, as well as implementing timeout triggers.

Test ELSP with embracing and representative real tests Even though we believe it to be very challenging, the theoretical support provided by this document should be further enhanced with real and representative tests on the protocol.

Improve compatibility The provided ELSP proof of concept was implemented with only Linux systems in mind, and makes use of several libraries not immediately available by all Linux distributions. As such, it would be beneficial to drop the usage of Nfqueue's and other libraries for the dynamic creation of tun interfaces and routing rules, as well as using raw packet programming.

Bibliography

[1] R. Abramov and A. Herzberg. "TCP Ack storm DoS attacks." In: *Computers & Security* 33 (2013), pp. 12–27.

[2] B. Carpenter and S. Brim. *Middleboxes: Taxonomy and issues*. Tech. rep. 2002.

[3] P. Eronen. "IKEv2 mobility and multihoming protocol (MOBIKE)." In: (2006).

[4] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. "RFC 6830: The Locator." In: *ID Separation Protocol (LISP)* (2013).

[5] D. Farinacci, D. Lewis, D. Meyer, and V. Fuller. "The locator/ID separation protocol (LISP)." In: (2013).

[6] M. Geva, A. Herzberg, and Y. Gev. "Bandwidth distributed denial of service: Attacks and defenses." In: *IEEE Security & Privacy* 12.1 (2014), pp. 54–61.

[7] N. W. Group et al. "Request for Comments (RFC) 4033,"" in: *Protocol Modifications for the DNS Security Extensions," Mar* (2005).

[8] S. Hanks, D. Meyer, D. Farinacci, and P. Traina. "Generic routing encapsulation (GRE)." In: (2000).

[9] D. Harkins and D. Carrel. *The internet key exchange (IKE)*. Tech. rep. 1998.

[10] K. Igoe, D. McGrew, and M. Salter. "Fundamental elliptic curve cryptography algorithms." In: (2011).

[11] S. Kent and K. Seo. "RFC-4301: Security Architecture for the Internet Protocol. 2005." In: *acesso em* 15.03 (2016).

[12] A. D. Keromytis, V. Misra, and D. Rubenstein. "SOS: Secure overlay services." In: *ACM SIGCOMM Computer Communication Review*. Vol. 32. 4. ACM. 2002, pp. 61–72.

[13] M. Komu, A. Keränen, and J. Melen. "Native NAT Traversal Mode for the Host Identity Protocol." In: (2017).

[14] L. Krämer, J. Krupp, D. Makita, T. Nishizoe, T. Koide, K. Yoshioka, and C. Rossow. "Amppot: Monitoring and defending against amplification ddos attacks." In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2015, pp. 615–636.

[15] T. Levä, M. Komu, A. Keränen, and S. Luukkainen. "Adoption barriers of network layer protocols: The case of host identity protocol." In: *Computer networks* 57.10 (2013), pp. 2218–2232.

[16] M. Luckie and B. Stasiewicz. "Measuring path MTU discovery behaviour." In: *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM. 2010, pp. 102–108.

[17] R Mahy, P Matthews, and J Rosenberg. "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN) RFC 5766." In: *Internet Society Request for Comments* (2010), p. 6.

[18] G. Maier, A. Feldmann, V. Paxson, and M. Allman. "On dominant characteristics of residential broadband internet traffic." In: *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*. ACM. 2009, pp. 90–102.

[19] R. P. Martins, J. L. Martins, and H. J. Domingos. "EIP-Preventing DDoD with Ephemeral IP Identifiers Cryptographically Generated." In: *arXiv preprint arXiv:1612.07065* (2016).

[20] J. Mirkovic and P. Reiher. "A taxonomy of DDoS attack and DDoS defense mechanisms." In: *ACM SIGCOMM Computer Communication Review* 34.2 (2004), pp. 39–53.

[21] P. Mockapetris. "RFC 1035—Domain names—implementation and specification, November 1987." In: *URL http://www. ietf. org/rfc/rfc1035. txt* (2004).

[22] R Moskowitz and R Hummen. "Hip diet exchange (dex)." In: *draft-moskowitz-hip-dex-00 (WiP), IETF* (2012).

[23] R. Moskowitz, T. Heer, P. Jokela, and T Henderson. *Host identity protocol version 2 (HIPv2)*. Tech. rep. 2015.

[24] P. Nikander, A. Gurtov, and T. R. Henderson. "Host identity protocol (HIP): Connectivity, mobility, multi-homing, security, and privacy over IPv4 and IPv6 networks." In: *IEEE Communications Surveys & Tutorials* 12.2 (2010), pp. 186–204.

[25] C. E. Perkins. "Mobile ip." In: *IEEE communications Magazine* 35.5 (1997), pp. 84–99.

[26] J. Postel et al. *Transmission control protocol RFC 793*. 1981.

[27] F. Quarter. "State of the Internet." In: *Security Report, Akamai Technologies https://www.akamai.com/us/e of-the-internet/q4-2017-state-of-the-internet-security-report.pdf. jsp. Accessed December* (2017).

[28] E. Rescorla. "Diffie-hellman key agreement method." In: (1999).

[29] C. Rossow. "Amplification Hell: Revisiting Network Protocols for DDoS Abuse." In: *NDSS*. 2014.

[30] T. Rozekrans, M. Mekking, and J. de Koning. "Defending against DNS reflection amplification attacks." In: *University of Amsterdam System & Network Engineering RP1* (2013).

[31] L. Rudman and B Irwin. "Characterization and analysis of NTP amplification based DDoS attacks." In: *Information Security for South Africa (ISSA), 2015*. IEEE. 2015, pp. 1–5.

[32] Y. B. Saied and A. Olivereau. "HIP Tiny Exchange (TEX): A distributed key exchange scheme for HIP-based Internet of Things." In: *Communications and Networking (ComNet), 2012 Third International Conference on*. IEEE. 2012, pp. 1–8.

[33] J. H. Saltzer, D. P. Reed, and D. D. Clark. "End-to-end arguments in system design." In: *ACM Transactions on Computer Systems (TOCS)* 2.4 (1984), pp. 277–288.

[34] J. J. Santanna, R. van Rijswijk-Deij, R. Hofstede, A. Sperotto, M. Wierbosch, L. Z. Granville, and A. Pras. "Booters—An analysis of DDoS-as-a-service attacks." In: *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE. 2015, pp. 243–251.

[35] D. Saucez, L. Iannone, O. Bonaventure, and D. Farinacci. "Designing a deployable internet: The locator/identifier separation protocol." In: *IEEE Internet Computing* 16.6 (2012), pp. 14–21.

[36] R. Sherwood, B. Bhattacharjee, and R. Braud. "Misbehaving TCP receivers can cause Internet-wide congestion collapse." In: *Proceedings of the 12th ACM conference on Computer and communications security*. ACM. 2005, pp. 383–392.

[37] M. Stiemerling. "NAT and firewall traversal issues of host identity protocol (HIP) communication." In: (2008).

[38] SYN Flood Attacks. *SYN Flood Attacks — Wikipedia, The Free Encyclopedia*. [Online; accessed 10-June-2017]. 2017. URL: https://en.wikipedia.org/wiki/SYN_flood.

[39] P. Vixie and V. Schryver. "Dns response rate limiting (dns rrl)." In: *URL: http://ss. vix. su/~ vixie/isc-tn-2012-1. txt* (2012).

[40] J. M. Wein, J. J. Kloninger, M. C. Nottingham, D. R. Karger, and P. A. Lisiecki. *Content delivery network (CDN) content server request handling mechanism with metadata framework support*. US Patent 7,240,100. 2007.

[41] L. Yong, E. Crabbe, X. Xu, and T. Herbert. *GRE-in-UDP Encapsulation*. Tech. rep. 2017.

[42] A. Young. "Connection-less Lightweight X. 500 Directory Access Protocol." In: (1995).

ANNEX I

Annex 1 - Code listing

Listing I.1: ELSP Daemon python proof of concept implementation for connectionless scenarios.

```
1  from netfilterqueue import NetfilterQueue
2  from scapy.all import *
3  import os
4  from array import array
5  import socket
6  import fcntl
7  import struct
8  import random
9  from Crypto.PublicKey import RSA
10 from Crypto.Hash import SHA256
11 from Crypto.Hash import HMAC
12 from Crypto.Random import random as cryptorandom
13 from Crypto.Signature import PKCS1_v1_5
14 from bitarray import bitarray
15
16 #The bellow iptable rules are for when previously known the peer.
17 #For multiple peers, use more broad rules.
18 iptableso = "iptables -I OUTPUT -d TARGET_IP/32 -j NFQUEUE --queue-num 1"
19 iptablesi = "iptables -I INPUT -s TARGET_IP/32 -j NFQUEUE --queue-num 1"
20
21 pendingAssociations = {}
22 associations = {}
23 nonELSPCompliant = []
24 puzzleK=16
25 hmacKey = b'testkey1'
26 rsaKey = RSA.generate(2048)
27 privKeyBin = rsaKey.exportKey('DER')
28 pubKeyBin = rsaKey.publickey().exportKey('DER')
```

```
29 privKeyObj = RSA.importKey(privKeyBin)
30 pubKeyObj = RSA.importKey(pubKeyBin)
31
32 print("Adding␣iptable␣rules␣:")
33 print(iptableso)
34 os.system(iptableso)
35 print(iptablesi)
36 os.system(iptablesi)
37
38
39 def get_ip_address(ifname):
40     s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
41     return socket.inet_ntoa(fcntl.ioctl(
42         s.fileno(),
43         0x8915,
44         struct.pack('256s', ifname[:15])
45     )[20:24])
46
47
48 def udp_send(dst, packet):
49     osocket.sendto(packet, (dst, 40000))
50
51
52 def callback(pkt):
53     payload= pkt.get_payload()
54     scpkt = scapy.all.IP(payload)
55     if(scpkt.src==THISIP):
56         outgoing(pkt)
57     else:
58         incoming(pkt)
59
60
61 def incoming(pkt):
62     payload= pkt.get_payload()
63     payloadArray = array('B', payload)
64     data = payloadArray[28:]
65     scpkt = scapy.all.IP(payload)
66     datastring="".join(map(chr, data))
67     if associations.has_key(scpkt.src) or pkt.get_mark()==123:
68         print('Received:␣', datastring)
69         pkt.accept()
70     else:
71         datarray = datastring.split(':')
72         stage = datarray[1]
73         nonce = datarray[0]
74         if (pendingAssociations.has_key(nonce)) and (stage!='I1'):
75             print('incoming␣stage:␣',stage)
76             if (getPreviousStage(stage) == pendingAssociations[nonce].split(':')[1]) and \
77                     (scpkt.src == pendingAssociations[nonce].split(':')[0]):
78                 if stage == 'R1':
```

```
79                      handleR1(scpkt.src, datarray, pkt)
80                  elif stage == 'I2':
81                      handleI2(scpkt.src, datarray, pkt)
82                  elif stage == 'R2':
83                      handleR2(scpkt.src, datarray, pkt)
84                  else:
85                      pkt.drop()
86              else:
87                  pkt.drop()
88          elif stage == 'I1':
89              print('incoming stage: ', stage)
90              handleI1(scpkt.src, datarray)
91          else:
92              pkt.drop()
93
94
95  def outgoing(pkt):
96      payload= pkt.get_payload()
97      payloadarray = array('B', payload)
98      ip_dstA = payloadarray[16:20]
99      ip_dst = str(ip_dstA[0]) + "." + str(ip_dstA[1]) + "." + str(ip_dstA[2])
100     + "." + str(ip_dstA[3])
101     if associations.has_key(ip_dst) or pkt.get_mark() == 123:
102         print('Sent: ', payload)
103         pkt.accept()
104     else:
105         data = payloadarray[28:]
106         datastring = "".join(map(chr, data))
107         datarray = datastring.split(':')
108         nonce = datarray[0]
109
110         if pendingAssociations.has_key(nonce):
111             if pendingAssociations[nonce].split(':')[0] != ip_dst:
112                 pkt.drop()
113             stage = datarray[1]
114             if stage == 'I1':
115                 pkt.accept()
116             elif stage == 'I2':
117                 pkt.accept()
118             elif stage == 'R1':
119                 pkt.accept()
120             elif stage == 'R2':
121                 pkt.accept()
122             print('outgoing stage: ', stage)
123         else:
124             newnonce = random.getrandbits(32)
125             message = str(newnonce)+':I1'
126             udp_send(ip_dst,message)
127             pendingAssociations[str(newnonce)] = ip_dst+':I1:'+datastring
128
```

```
129
130 def handleI1(ip_dst, datarray):
131     #availability function would be here deciding type of response
132
133     hmac = HMAC.new(hmacKey)
134     rndfile = cryptorandom.Random.new()
135     i=rndfile.read(256)
136     i=bin2hex(i)
137
138     nonce = datarray[0]
139     hmac.update(b''+str(nonce)+str(ip_dst)+str(i)+str(puzzleK))
140     hmacdigest = hmac.hexdigest()
141     hmacdigest = bin2hex(hmacdigest)
142
143     toHash = str(nonce)+str(hmacdigest)
144     hash = SHA256.new()
145     hash.update(toHash)
146
147     signer = PKCS1_v1_5.new(privKeyObj)
148     signed = signer.sign(hash)
149     signed = bin2hex(signed)
150
151     message = str(nonce)+':R1:'+str(bin2hex(pubKeyBin))+":"+i+':'+str(puzzleK)+':'
152     +str(hmacdigest)+':'+str(signed)
153     pendingAssociations[nonce]=str(ip_dst)+':R1'
154     udp_send(ip_dst, message)
155
156
157 def handleR1(ip_dst, datarray, pkt):
158     nonce = datarray[0]
159     outerPubKeyBin = datarray[2]
160     i=datarray[3]
161     i=hex2bin(i)
162     k=int(datarray[4])
163     hmac=datarray[5]
164     signature=datarray[6]
165
166     toVerify = str(nonce)+str(hmac)
167     hash = SHA256.new()
168     hash.update(toVerify)
169
170     outerPubKeyObj = RSA.importKey(hex2bin(outerPubKeyBin))
171     verifier = PKCS1_v1_5.new(outerPubKeyObj)
172     verified = verifier.verify(hash, hex2bin(signature))
173
174     if verified == False:
175         print('Wrong signature received')
176         pkt.drop()
177     rndfile = cryptorandom.Random.new()
178     contador = 0
```

```
179     print('Starting to solve puzzle')
180     while True:
181         contador += 1
182         j = rndfile.read(256)
183         solution=tryPuzzleSolution(i, ip_dst, nonce, k, j)
184         if solution != None:
185             break
186 
187     print('Puzzled solved, tries needed: ', contador)
188     print('Solution: ', solution)
189     request = pendingAssociations[nonce].split(':')[2]
190     message = str(nonce)+':I2:'+str(bin2hex(i))+':'+str(k)+':'+str(solution)+':'
191     +str(hmac)+':'+str(request)
192     pendingAssociations[nonce] = str(ip_dst)+':I2:'+request+':'+outerPubKeyBin
193     udp_send(ip_dst, message)
194 
195 
196 def handleI2(ip_dst, datarray, pkt):
197     nonce = datarray[0]
198     i = datarray[2]
199     k = int(datarray[3])
200     solution = datarray[4]
201     solution = hex2bin(solution)
202     hmacreceived = datarray[5]
203     request = datarray[6]
204 
205     hmac = HMAC.new(hmacKey)
206     hmac.update(b''+str(nonce)+str(ip_dst)+str(i)+str(puzzleK))
207     hmacdigest = hmac.hexdigest()
208     hmacdigest = bin2hex(hmacdigest)
209     i = hex2bin(i)
210 
211     if hmacdigest!=hmacreceived:
212         print('hmac errado')
213         pkt.drop()
214     triedsolution = tryPuzzleSolution(i, THISIP, nonce, k, solution)
215     if bin2hex(solution) != triedsolution:
216         print('puzzle errado')
217         pkt.drop()
218     else:
219         print('puzzle certo')
220         associations[ip_dst] = nonce
221 
222         #change current packet to have original request to avoid another packet exchange
223         temppacket = scapy.all.IP(pkt.get_payload())
224 
225         beforelen = len(temppacket[scapy.all.UDP].payload)
226         temppacket[Raw].load = request
227         afterlen = len(temppacket[scapy.all.UDP].payload)
228         dif = afterlen-beforelen
```

```
229
230         temppacket[scapy.all.UDP].len = temppacket[scapy.all.UDP].len+dif
231         temppacket[scapy.all.IP].len = temppacket[scapy.all.IP].len+dif
232
233         del temppacket[scapy.all.UDP].chksum
234         del temppacket.chksum
235         temppacket.show2()
236         pkt.set_payload(str(temppacket))
237         pkt.set_mark(123)
238
239         toHash = str(nonce)
240         hash = SHA256.new()
241         hash.update(toHash)
242
243         signer = PKCS1_v1_5.new(privKeyObj)
244         signed = signer.sign(hash)
245         signed = bin2hex(signed)
246
247         message = str(nonce)+':R2:'+signed
248
249         udp_send(ip_dst, message)
250
251         pkt.accept()
252
253
254 def handleR2(ip_dst, datarray, pkt):
255     nonce = datarray[0]
256     signature = datarray[2]
257
258     toVerify = str(nonce)
259     hash = SHA256.new()
260     hash.update(toVerify)
261     outerPubKeyObj = RSA.importKey(hex2bin(pendingAssociations[nonce].split(':')[3]))
262     verifier = PKCS1_v1_5.new(outerPubKeyObj)
263     verified = verifier.verify(hash,hex2bin(signature))
264
265     if verified == False:
266         print('Received invalid signature')
267         pkt.drop()
268     else:
269         print('Association fully complete!!!')
270         associations[ip_dst]=nonce
271
272
273 def tryPuzzleSolution(i,ip_dst,nonce,k,j):
274     counter = 0
275
276     sha = SHA256.new(i + j + ip_dst + ip_dst + nonce).digest()
277     bl = tobits(sha)
278     lenght = len(bl)
```

```
279
280     for x in range(lenght - k, lenght):
281         if bl[x] == 0:
282             counter += 1
283     if counter == k:
284         solution = j
285         return bin2hex(solution)
286     else:
287         return None
288
289
290 def tobits(s):
291     ba = bitarray()
292     ba.frombytes(s)
293     return ba.tolist()
294
295
296 def getPreviousStage(stage):
297     if stage=='R1':
298         return 'I1'
299     elif stage == 'I2':
300         return 'R1'
301     elif stage == 'R2':
302         return 'I2'
303
304
305 def hex2bin(hex):
306     return binascii.unhexlify(hex)
307
308
309 def bin2hex(bin):
310     return binascii.hexlify(bin)
311
312
313 THISIP = get_ip_address(INTERFACE_NAME)
314 nfqueue = NetfilterQueue()
315 nfqueue.bind(1, callback)
316 osocket=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
317
318 try:
319     nfqueue.run()
320
321 except KeyboardInterrupt:
322     print('')
323     nfqueue.unbind()
324     os.system('iptables␣-F')
```