University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Mitigation as a Service in a Cooperative Network Defense

Stephan Mannhart
Steinach, Switzerland
Student ID: 11-917-515

Supervisor: Bruno B. Rodrigues, Eder Scheid
Date of Submission: July 31, 2018

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Das ständig wachsende Internet umfasst eine Vielzahl an verbundenen Geräten, die entweder selbst Opfer eines Distributed Denial of Service (DDoS) Angriffs im grossen Stil werden oder direkt Teil der Angriffswelle sind. Die Abwehr solcher Attacken erfordert die Zusammenarbeit verschiedener autonomer Systeme um eine schnelle Abwehr der Attacke direkt an der Quelle zu ermöglichen. Diese Zusammenarbeit hilft, Bandbreite und Computerressourcen jedes einzelnen Internetbenutzers und speziell der Opfer dieser Angriffe zu schützen [48].

Diese Masterarbeit beschreibt im Detail das Design und die Implementation eines modularisierten, Netzwerk-agnostischen und kooperativen DDoS Abwehrsystems, bestehend aus mehreren dezentralisierten Instanzen, die in Zusammenarbeit Attacken gegen jedes Mitglied dieser Kooperative abwehren. Der neuartige Aspekt dieses Abwehrsystems, das mehrere Domänen überspannt, ist die Verwendung der Ethereum Blockchain zur Signalübertragung, um alle Mitglieder der Kooperative über laufende Angriffe zu informieren. Die Implementation ist ein Re-Engineering des Proof of Concept Blockchain Signaling Systems (BloSS) entwickelt von Rodrigues *et al.* [46] mit dem Ziel, eine flexiblere Plattform zu entwickeln, die als Fundament für zukünftige Forschungsarbeiten in Richtung eines vollautomatisierten Mitigation-as-a-Service (MaaS) Systems dienen soll. Zusammen mit dem Re-Engineering des BloSS werden Ansätze für eine Proof-of-Mitigation Lösung präsentiert, mit dem Ziel, ein zahlungsbasiertes Anreizsystem zu ermöglichen, welches in das MaaS System integriert werden kann.

Die gesamte Entwicklung wird auf einem Demonstrationssystem basierend auf Einplatinenrechnern und Software-Defined Networking (SDN) Hardware ausgewertet. Als Resultat dieser Auswertungen wird die kurze Reaktionszeit zur kollaborativen Abwehr einer DDoS Attacke aufgezeigt, sowie die ressourcenschonende Implementation des Verschlüsselungssystems beleuchtet, welches die Vertraulichkeit und Richtigkeit der Attackinformationen in der Blockchain gewährleistet.

The ever-growing Internet brings with it a multitude of connected devices that could either fall victim to large-scale Distributed Denial of Service (DDoS) attacks or be part of the attack themselves. Defending against these attacks requires collaboration among different autonomous systems (ASes) to enable a swift mitigation of the threat directly at the source. This helps to save bandwidth as well as computing resources for the internet as a whole and the victims of these attacks in particular [48].

This thesis details the design and implementation of a modularized, network-agnostic and cooperative DDoS defense system consisting of multiple, decentralized instances working together to mitigate attacks targeted at any member of this alliance. The novel aspect of this multi-domain defense system is the use of the Ethereum blockchain as a signaling medium to inform all parties in the system about ongoing attacks. The implementation is a re-engineering of the proof of concept Blockchain Signaling System (BloSS) developed by Rodrigues *et al.* [46] with the goal to build a more flexible platform that can serve as the basis for future research toward realizing a fully automated Mitigation-as-a-Service (MaaS) offering. Together with the re-engineering of the BloSS, approaches to provide a Proof-of-Mitigation are presented, with the aim to enable payment-based incentive schemes to be integrated into the aforementioned MaaS.

The entire development effort is evaluated on a demonstration system based on single board compute nodes and Software-Defined Networking (SDN) hardware. The evaluation highlights the short delay in blocking multi-domain attacks and the minimal footprint of the implemented encryption scheme that provides confidentiality and integrity for the attack information published on the blockchain.

Acknowledgments

I am using this opportunity to thank my advisors Bruno Rodrigues and Eder Scheid for their support and guidance throughout the thesis. A special thanks goes to Bruno Rodrigues for being available around the clock to answer my questions and troubleshoot any obstacles I encountered while working on the BloSS.

Last but not least I thank my family for allowing me to pursue my thesis in a supportive environment and motivating me along the way.

Contents

Chapter 1

Introduction

The growing number of devices in the modern internet leads to a multitude of problems. One of them are large-scale distributed denial of service (DDoS) attacks consuming ever-increasing volumes of bandwidth for all internet users and critical amounts of computing resources for the targets of these attacks [48]. Mitigating DDoS attacks therefore becomes an important task to ensure that the highly connected systems in use today are guaranteed to work without major interruptions for many more years to come.

1.1 Motivation

DDoS attacks either focus on overloading the target's networking infrastructure by bombarding them with high-bandwidth traffic or they directly target the computing resources through malformed packets designed to generate overwhelming computational loads [27]. In both cases, defending against these attacks at the ingress point of the traffic quickly becomes infeasible due to the constrained resources available for defense at the autonomous system (AS) being targeted. In contrast to this centralized approach, a decentralized defense, where the DDoS mitigation takes place at the egress of the attack on the individual ASes is preferable [58].

The main challenge of such a decentralized defense scenario is the additional communication and coordination overhead [58]. This is caused by the need to keep all the ASes in sync in regard to the attack taking place to enable a cooperative effort toward mitigating the attack collaboratively. Communication also needs to go through a trusted channel [58] to make sure that the attack information provided to all ASes in the distributed defense is reliable. Providing the necessary communication means to this alliance of ASes for cooperative defense purposes is however not enough to bolster the willingness among all participants to help each other. This cannot simply be based on goodwill but needs to be incentivized, for example through monetary reimbursements [58].

Incentives among the members of the cooperative defense are however not the only expense that needs to be covered. Costs in the form of CAPital EXpenditures (CAPEX) for setting up and maintaining the communication infrastructure as well as OPErating

EXpenditures (OPEX) to cover resource utilization costs for the actual attack mitigation need to be taken into consideration to enable a successful cooperative defense alliance [58]. By introducing a service model, these costs can be shifted from the AS operators to potential customers subscribing to a purpose-built Mitigation-as-a-Service (MaaS) offering [23]. By subscribing to such a service, a customer will receive protection from DDoS attacks emanating from within the DDoS alliance. This works by directly using the fees paid by the customer as incentives between the operator of the AS the customer resides in and the operators of the ASes where the attack stems from.

The main problem in regards to incentivized cooperative defense is however not the sourcing of the incentives but to ensure a smooth exchange of mitigation services to receive incentives without being able to rely on underlying trust between the parties or a reputation and reward scheme substituting the trust. Such an exchange can only be successful, if a verifiable proof for mitigation service completion can be provided since that would allow a direct payout of the reward upon completion of the mitigation without inherent trust between the involved parties.

1.2 Thesis Goals

This thesis consists of an engineering-, as well as a research-focused part.

The engineering part consists of developing a stand-alone cooperative defense solution based on the Blockchain Signaling System (BloSS) [46] by extending the existing proof of concept implementation of the BloSS toward a network-infrastructure-agnostic and modularized defense system. This modularized version of the BloSS serves the purpose of providing a testbed toward a full MaaS offering.

In addition to the engineering-focused development of the modularized BloSS, the research question of providing a mitigation proof is discussed by presenting four approaches and conducting a qualitative evaluation of each approach.

The overall goal of this work is to advance the research in the field of cooperative DDoS defenses by addressing two main aspects: Creating an easy to deploy platform to conduct the mitigation and providing concrete approaches to efficiently automate a mitigation service through an independently verifiable mitigation proof.

1.3 Thesis Outline

Chapter 2 surveys existing cooperative defense solutions to support informed decisions in regards to the design and development of the modularized BloSS. This chapter is largely based on literature research conducted throughout the first part of the thesis.

Chapter 3 details the research part of the thesis with the goal of enabling an efficient and fully automated incentive scheme based on a Proof-of-Mitigation that unambiguously and

reproducibly asserts the completion of a mitigation service in the context of a cooperative defense system.

Chapter 4 presents the design of the modularized BloSS. This chapter provides a high-level overview about the concepts involved in the BloSS and how the whole system has been re-engineered toward a more flexible, modularized system capable of being adapted to different networking architectures and deployment scenarios.

Chapter 5 serves as a technical documentation of the BloSS including the test setup in the form of a single board computer cluster running a demonstration installation of the entire system.

Chapter 6 evaluates the re-engineered BloSS in terms of performance both with and without encrypted data exchange.

Chapter 7 discussed the evaluation results in regards to practicability and performance of the modularized BloSS.

Finally, **Chapter 8** concludes this thesis with final considerations and pointers to future research enabling a full-fledged MaaS offering.

Chapter 2

Related Work

This chapter provides an overview about existing defense systems to counter network- and transport-level DDoS attacks both from a centralized as well as a decentralized perspective. A summary of all systems is provided at the end of the chapter to provide a condensed review highlighting advantages as well as disadvantages.

2.1 Centralized DDoS Defense Mechanisms

The traditional solution to defend against DDoS attacks revolves mainly around two perspectives: Source- as well as destination-based mitigation [7]. In the case of source-based mitigation, the defense system tries to identify malicious outgoing traffic at an AS and blocking the attack traffic upon detection [58]. In contrast to this, destination-based mitigation analyzes incoming traffic at the edge routers and access routers of the attack-target AS with the goal of filtering the traffic before it reaches the AS [58].

Both perspectives have in common that they only consider individual, isolated ASes, either on the source or destination side of the attack. This gives them the characteristic of a centralized defense approach, since no coordination among the ASes involved in the overall attack exists. The classification as a centralized defense approach does however not mean, that only a centralized instance of the system exists. Especially for the source-based approaches, multiple instances of the defense system are installed on as many ASes as possible to counter the funnel-effect of DDoS attacks as demonstrated by Zargar *et al.* [58]: DDoS attacks start out at a multitude of origins and converge increasingly toward a single attack target. By analyzing traffic at multiple sources, the traffic- and resource-multiplication of the DDoS attack throughout the funnel can already be mitigated at its origin, preventing excessive loads at the attack target. Destination-based approaches are more constrained in regards to the number of instances deployed, however it is important to realize that once again, centralized destination-based approaches still have to be present on each AS requiring DDoS protection. There is no central instance capable of handling DDoS protection for multiple ASes at the same time.

2.1.1 Source-Based Mechanisms

The following exemplary mechanisms are all source-based approaches with the goal of decreasing the attack footprint at its origin. Advantages of these mechanisms are the low amount of traffic required to be analyzed, the ability to stop an attack right at the source, easy tracing of the attack back to its origin and the ability to dedicate more resources toward the mitigation efforts since the attack itself does not consume a lot of resources due to its small footprint [27].

The low amounts of attack-bandwidth present at the source routers can however also be seen as a disadvantage since it is much harder to differentiate between good and bad traffic if the traffic volume cannot be used as an indicator. Since DDoS attacks are by nature massively distributed, the attack bandwidths will most likely be very small at the individual source routers and without any coordination among the routers, they might all decide to ignore the attack altogether. Since the entire mitigation endeavor also does not directly contribute toward increased performance or stability of the AS deploying the defense mechanism, motivation to run such a system is minimal [58].

2.1.2 DDoS netWork Attack Recognition and Defense (D-WARD)

D-WARD is a source-based defense strategy which monitors the outgoing traffic and compares it to predefined traffic models which helps the system decide whether a specific flow includes malicious traffic [27]. The D-WARD system is installed on multiple systems, however there exists no coordination among these systems, which clearly makes D-WARD a centralized approach [58].

As soon as a network flow being monitored by D-WARD is detected to be of malicious origin, the traffic is rate-limited to stifle the effectiveness of the attack [27]. Rate-limiting the flows allows ongoing monitoring to be able to lift the rate limit as soon as the flow ceases to exhibit malicious characteristics. This is preferable to outright stopping the flows since misclassifications cannot be detected if continuous monitoring of the respective flows is unavailable.

Detection works by monitoring the behavior of a flow to detect clear indicators of communication problems in the traffic such as ”a reduction in the number of response packets or longer inter-arrival times” [27]. These indicators are compiled into a model which is used to analyze statistics obtained from the traffic ingressing and egressing the source router at the AS. As soon as a match occurs between the predefined model and the recorded traffic, the discussed rate limiting is applied which is then continuously adjusted based on whether the traffic keeps exhibiting the characteristics defined in the model [27].

2.1.3 MUlti-Level Tree for Online Packet Statistics (MULTOPS)

The MULTOPS data structure pursues a similar strategy as D-WARD by analyzing specific characteristics of the ingressing and egressing traffic of an AS. Instead of gathering

statistics and trying to find patterns that match a predefined model, MULTOPS is an attack-resistant data structure storing statistics about aggregated traffic to and from the AS it is running on. These statistics are then used to find significant differences between the amount of incoming and outgoing traffic which would indicate an attack going in one direction. This is based on the assumption that benign traffic exhibits symmetric characteristics for the amount of incoming and outgoing traffic since packets from one host are generally acknowledged by the other host [11].

MULTOPS is essentially a tree with four levels where the root node contains aggregate traffic rates to and from network hosts managed by the AS [11]. The leaves in the tree represent individual traffic flows for individual network hosts. If the traffic rate for a packet-flow exceeds a specified threshold, a new node in the MULTOPS tree is created. This continues until the leaves on level 4 of the tree are reached and the flow is associated with the specific destination host in the leaf. The Nodes are subsequently destroyed if the traffic falls below the threshold. If the traffic continues to exceed the threshold, the source router at the AS is instructed to drop the packets originating from within the ASes domain. [11].

Gil *et al.* [11] implemented MULTOPS as a module of the Click router [19]. The advantage of the Click router is the ability to chain multiple modules, so called "elements"[19] together to form a processing pipeline for incoming packets. Each element only performs very simple functions such as "communicating with devices, queuing packets, and implementing a dropping policy"[11]. The MULTOPS implementation contains two elements called "IPRateMonitor"[11] and "RatioBlocker"[11] which are directly interconnected in a way that the rate of traffic for incoming and outgoing packets is monitored. Based on these monitored rates, packets are dropped if the ratio is out of balance according to the predefined thresholds.

2.1.4 Destination-Based Mechanisms

Destination-based approaches toward DDoS mitigation face a big challenge in the form of the high bandwidth of the attack traffic arriving at the routers they are managing. This leads to high resource usages on the mitigation system due to the overhead of analyzing the attack traffic not at its origin but only very near to its destination [58]. Since attack traffic is left untouched throughout its preceding route, resources are wasted which could have been saved if the traffic would have been classified before [58].

The advantage of the late response to the attack traffic is that it is much easier to identify malicious traffic simply based on its volume and destination[58]. Another clear advantage are the direct incentives for the operator of destination-based mitigation systems: By running such a system, they are directly able to protect hosts within their AS from attacks, which represents an added value to all or their users [58].

Following defense mechanisms demonstrate different approaches to mitigate DDoS attacks in a destination-based manner. These mechanisms take place at the edge routers or access routers of the AS deploying them [58].

2.1.5 IP Traceback

Destination-based mechanisms are in the unique position to be able to analyze the entire attack traffic since they are very close to the attack target. This can be used to trace back the attack origins to better identify malicious traffic and easily differentiate it from normal traffic. Tracing back attack traffic to its origin is however not a straight forward process, since attackers often spoof their IP addresses, making it much harder to find the actual origin [18].

Traceback approaches are divided into preventive and reactive approaches [18]. The less common preventive approaches try to block packets originating from spoofed IP addresses as a direct measure against DDoS attacks [18]. This is accomplished by examining every packet arriving at the edge or access router of an AS which can become quite resource-intensive with increasing volume of a DDoS attack.

Reactive approaches on the other hand can generally be divided into link testing and packet marking schemes [18]. Link testing tries to identify the link which was used to transfer the attack packet. This is continued upstream until the source is reached, which gives a clear indication where a specific attack came from [18]. Instead of manually testing all the links right up to the attackers, packet marking allows to mark packets on the router while forwarding them [18]. The victim can later observe the path of a packet by examining the inserted marks [18]. Even though this scheme requires less work from the attack victim to determine the source of the attack, compatible routers are required that are able to mark the packets on their way.

Traceback approaches generally suffer from deployment and operational challenges due to the complexity of these schemes: Preventively examining each packet or marking packets for later examination can be automated, however it requires additional resources in the form of computing power for examination and compatible routing hardware for packet marking [58]. In the case of packet marking, different paths might receive the same mark, which increases the false positive rate dramatically [58].

2.1.6 Analyzing Management Information Base (MIB)

MIB data contains statistical variables about the traffic in a Network Management System (NMS) [4]. These variables are organized in groups such as ip, icmp, tcp, udp and snmp, generally grouping the variables with the corresponding protocols [4]. The goal in utilizing the MIB for destination-based DDoS prevention is to deduce patterns prior to a DDoS attack based on the preparations taken by the attacker to instruct the slave machines utilized in a DDoS attack [4]. This represents a much more efficient approach toward DDoS mitigation especially considering the high volume of attack traffic destination-based approaches normally need to defend against. By setting up such an online monitoring solution in the form of an Intrusion Detection System (IDS) for DDoS attacks, blocking of attackers can already occur throughout the preparation stages of the attack instead of after the attack is already taking place and wasting resources [4].

This scheme can already be applied for destination-based mitigation where the NMS defending against the attack only manages the DDoS target. If attackers are however also part of the managed domain, additional instructive traffic between the attack master and the attack slaves can be analyzed to generate pointers for an early warning system [4]. The proactive detection of attacks works in three steps [4]:

1. The first step is concerned with collecting information about possible MIB variables suitable to act as precursors for an imminent attack at a target host. These variables can be found by analyzing traffic data from past attacks and determining patterns based on the available MIB variables.

2. If attack hosts are within the domain of the NMS being protected, correlations between MIB variables for the attack hosts and the variables for the target hosts from step 1 can be calculated.

3. These correlations can then be used to increase the significance of the precursors found in step 1 to be able to react to imminent attacks even faster.

2.2 Decentralized DDoS Defense Mechanisms

The source- and destination-based approaches employed by the centralized DDoS defense systems fall short in regards to the highly distributed nature of the attacks. Source-based approaches such as D-WARD discussed in Section 2.1.2 or MULTOPS presented in Section 2.1.3 often fail to identify the attack as a whole and only see parts of it. Destination-based approaches on the other hand are able to detect the entire attack volume, however this quickly turns into their biggest weakness since they then have to mitigate massive amounts of traffic at once. Preventive measures such as utilizing MIB for attack precursor detection as seen in Section 2.1.6 can help in that regard, however the resources required to mitigate the attack are still prohibitively high.

A hybrid approach, combining source- and destination-based approaches as well as introducing a degree of cooperation among the defense systems, is able to better cope with these highly distributed attacks [58]. By allowing the individual defense systems to communicate among each other, attack information can be shared to be able to react to attackers that might have remained undetected on a source system whereas a destination system might have identified the attack affecting one of the hosts within their domain.

2.2.1 IETF DOTS

The Internet Engineering Task Force (IETF) Distributed-Denial-of-Service Open Threat Signaling (DOTS) architecture [28] was devised as a standardization attempt for collaborative DDoS defense. Through the DOTS protocol, data models are provided to enable intra- and inter-organizational DDoS defense with multiple parties [34]. DOTS specifically focuses on aiding with the coordination of attack responses with a client and server

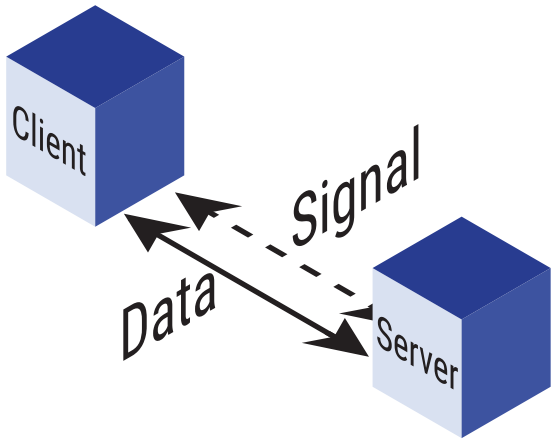

Figure 2.1: DOTS server and client model with data and signal channel

model. The DOTS client requests mitigation from the DOTS server after detecting an ongoing attack [28].

Communication between the DOTS server and client takes place over a data- as well as a signal-channel as illustrated in Figure 2.1. The signal channel is used by the client to request mitigation from the server and the server uses the signal channel in turn to inform the client about the status of the mitigation [28]. As part of the information provided by the client to signal the server for help, attack targets as well as telemetry data about the attack can be provided through the signal channel to simplify the mitigation for the server [28].

The data channel, which is an optional component in the DOTS scheme, is used to exchange additional configuration information that can then be used in addition to the information transferred through the signaling channel. These configurations may consist of host identifiers, blacklists, whitelists, traffic filters or DOTS client provisioning [28].

During operation, the DOTS system differentiates between manual and automatic mitigation requests [28]. Manual requests represent a labor-intensive way of requesting mitigation from a DOTS server by submitting the request through a text prompt or a graphical user interface [28]. The requests can be received by the DOTS client operator through channels such as ”phone, e-mail, Web-portal, etc.” [34] and then manually relayed to the server. This request method is very slow since the operator as a middleman needs too much time to receive, verify and relay new requests. The automatic approach outlined in [28] allows for fast mitigation by directly triggering requests to the DOTS server upon fulfilling pre-defined network conditions [28].

To extend the reach of the mitigation system beyond the current domain served by the most intermediate DOTS server, recursive signaling [28] can be used. The recursive part of this signaling scheme is represented by an additional DOTS client operated by a DOTS server operator which can be used to signal other domains to enable true cooperative defense as illustrated in Figure 2.2.

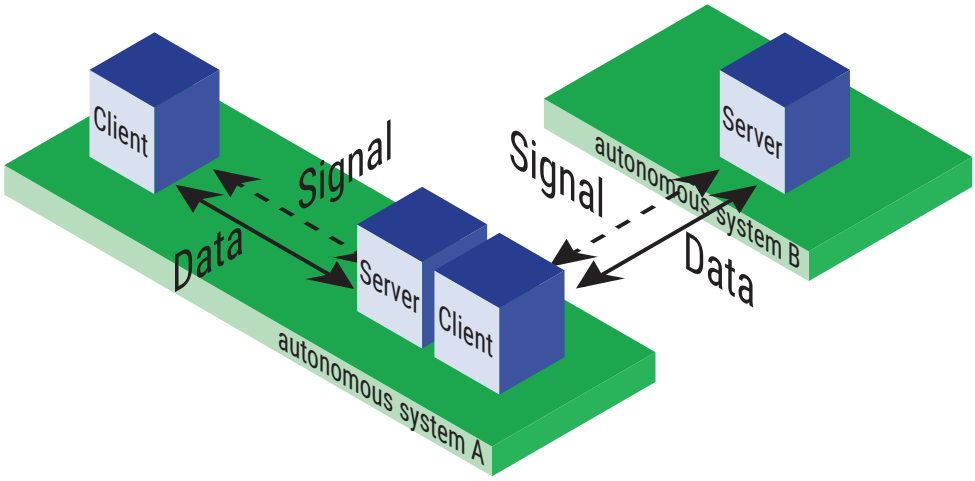

Figure 2.2: Recursive signaling in DOTS

2.2.2 CoFence

The inter-domain mitigation requires an efficient scheme to provision the filters required to cooperatively defend against a large-scale DDoS attack. CoFence presented by Rashidi *et al.* [44] addresses this problem by relaying attack traffic to other participants in the cooperative alliance. This circumvents the problem of filter provisioning and provides a simple check for the efficacy of the mitigation since the participating systems will only relay back attack-free traffic which directly proves that they have actually conducted the mitigation [44].

Deployment of a CoFence instance relies on Network Function Virtualization (NFV)[32] to simplify the instantiation of the entire system. Utilizing NFV for a collaborative defense system can help to persuade potential new members of a DDoS alliance to join, since device upgrading and creation is very fast and low cost due to the virtualized nature of all networking components [44]. Instead of relying on fixed hardware-based networking solutions, commodity hardware can be used to launch virtualized networking appliances that can be spun up on demand [9].

2.2.3 Bohatei

The emerging paradigm of NFV is often used in conjunction with Software-Defined Networking (SDN). Through SDN, the data plane is decoupled from the control plane of the networking infrastructure, allowing tailored solutions for specific networking needs [9]. The routing required to relay the attack traffic in the case of CoFence could be simplified through SDN-based networking as similar mitigation solutions such as Bohatei, presented by Fayaz *et al.* [9] demonstrate.

Bohatei serves as a clear indicator of the scalability advantages in using SDN- and NFV-based networking to tackle the DDoS defense problem. The Bohatei proof of concept

implementation discussed in [9] is realized with the OpenDaylight SDN controller[38] together with an assortment of Open Source tools to facilitate routing and mitigation such as OpenvSwitch[42], Snort[5], Bro[39] and iptables[43]. Experiments conducted by Fayaz *et al.* show, that the Bohatei solution was able to mitigate attacks with a total throughput of up to 100 Gbps while only requiring hardware which was 2.1 to 5.4 times more cost-effective compared to "fixed defense facilities" [9]

The proof-of-concept implementation of Bohatei as presented by Fayaz *et al.* does not directly incorporate inter-domain DDoS defense, however employing a scheme such as Pushback [16], which allows edge and access routers to relay traffic filtering to routers further upstream, could expand the approach and allow multi-domain cooperative defense with the flexibility of SDN and NFV.

2.2.4 DefCOM

The highly distributed nature of DDoS attacks requires an equally distributed defense system. This is where DefCOM comes in: It uses existing defense strategies and provides a framework in the form of a cooperative overlay network to exchange mitigation information across multiple domains [37].

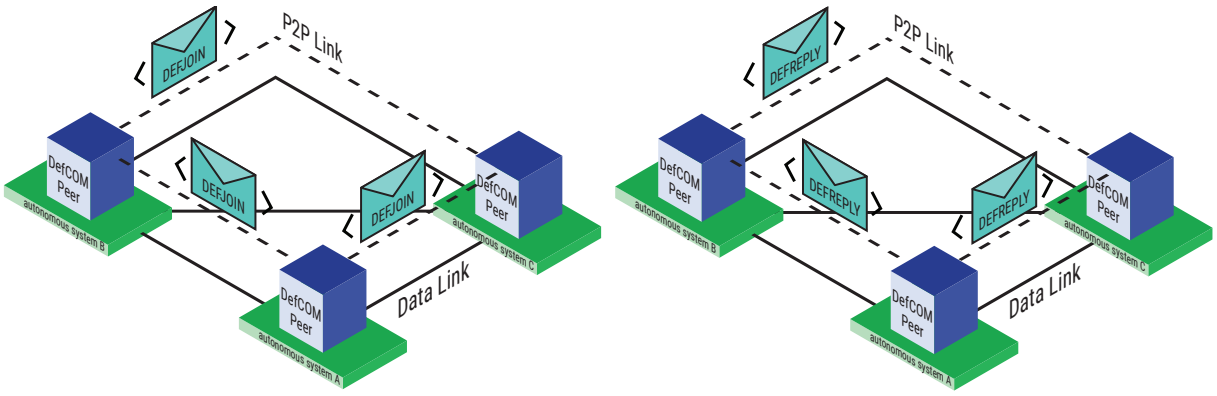

Figure 2.3: Formation of the DefCOM overlay network

DefCOM is specifically geared toward protection against flooding DDoS attacks and focuses on three critical defense functionalities [37]:

1. Differentiating between benign and attack traffic through traffic classification
2. Rate limiting attack traffic to free resources
3. Alert generation to signal all members of the cooperative defense about the IP address of the attack target as well as rate limits required to resolve traffic bottlenecks for the attack target

The strength of the DefCOM system lies in the highly distributed and scalable overlay network used for communication. The overlay network is however only used for control messages, data packets still travel on the data links defined by the underlying routing protocols [37]. Every member of the DefCOM network is known as a ”peer” [37] and formation of the entire overlay network is accomplished with the following two steps as illustrated in Figure 2.3:

1. A peer starts the formation by flooding a currently unassigned UDP port with DEFJOIN messages. These messages serve the task of inviting other peers to join the overlay network but also contain a session key to enable encryption for future control messages.

2. Peers receiving these DEFJOIN message are able to extract certificate information to verify the authenticity of the message and answer the invitation with a DEFREPLY message

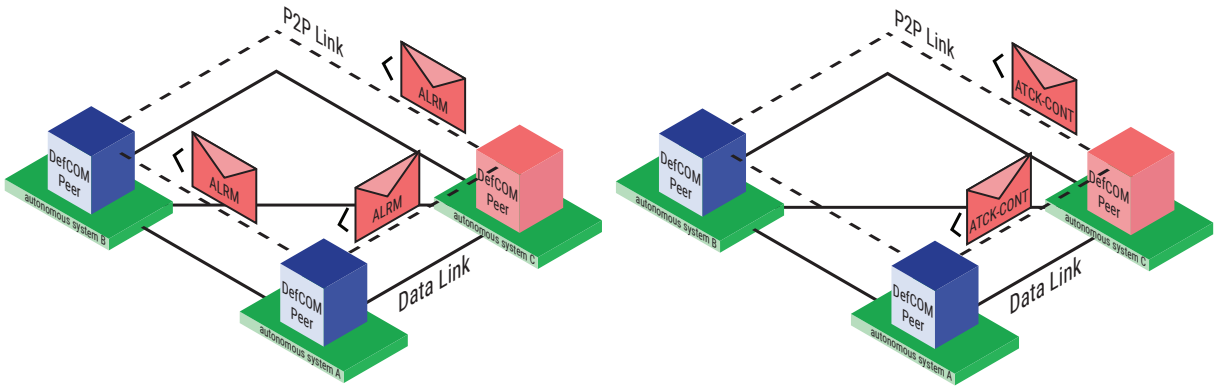

Figure 2.4: Alarm propagation in the DefCOM overlay network

As soon as the overlay network is formed, attack mitigation can be initiated. This is accomplished by sending out an ALRM message to all connected peers as illustrated in Figure 2.4. The routing structure from the source of the attack to the destination can subsequently be deduced by examining packet markings to establish parent-child relationships based on the volume and direction of the attack traffic. Rate limiters along the deduced routing structure can then start reducing the throughput of attack traffic. Each node being affected by the rate-limiting sends out ATCK-CONT messages to let other peers know that the attack is still ongoing. As soon as no more ATCK-CONT messages are received by any of the peers for a pre-specified amount of time, the rate limiters can cease their operation [37].

The implementation of DefCOM presented by Oikonomou *et al.* [37] is based on a RedHat 9.0 router running all DDoS defense components as Linux kernel modules [37]. For traffic classification purposes, they used D-WARD discussed in Section 2.1.2 but deactivated all attack mitigation functionality of D-WARD to reduce it to a pure traffic classification

engine. By utilizing the existing source-based D-WARD classifier and implementing it in the context of an overlay network, communication among multiple peers can be leveraged to make the overall mitigation system much more efficient compared to the centralized approach discussed in Section 2.1.2.

2.3 Systems Overview

Table 2.1: Overview of DDoS defense systems

		Details	Advantages	Disadvantages
Centralized	**D-WARD**	A source-based system that compares outgoing traffic to predefined models to identify malicious traffic. Rate-limiting is used to mitigate the attack.	The attack is mitigated close to the target. Rate-limiting allows re-examination of attack traffic. The predefined model provides a comprehensive rule set.	Individual D-WARD installations do not communicate. The attack is harder to identify at the source without collaboration.
	MULTOPS	A source-based system that compares the ratio between incoming and outgoing traffic to find imbalances hinting at an attack. Identified attack packets are subsequently blocked	The implementation based on the Click router is simple to deploy and easy to maintain. The underlying MULTOPS data structure guarantees efficient handling of large traffic volumes.	The MULTOPS data structures are not synced among multiple instances, leading to duplication. Specific software in the form of the Click router is required to run the system. Attack identification without communication is harder at the source.
	IP Traceback	A destination-based system divided into preventive traceback, which directly blocks packets with spoofed addresses and reactive traceback, which tries to identify the path leading to the attacker to specifically block packets originating from there.	Narrowing down the origin of the attack allows for tailored defense since packets originating from a known attack path can directly be blocked.	High volume attacks can quickly overwhelm individual packet inspection, making the preventive approach impractical. The reactive approach requires additional routing hardware to enable packet marking and exhibits high complexity in regards to link testing.
	Analyzing MIB	A destination-based system to analyze communication between an attack master and the multitude of attack slaves to find specific patterns that can be observed by analyzing MIB variables.	The "Intrusion Detection System for DDoS" solves the bottleneck problem of traditional destination-based approaches by already blocking attack traffic before the overall attack starts. Analysis of the MIB in preparation for this preventive measure is also possible on collected, offline attack data.	The collected patterns could be used together with a source-based approach, however that would require communication. Analyzing old attack data might not reveal patterns relevant to upcoming attacks.
Decentralized	**IETF DOTS**	DOTS proposes a client server model where clients request mitigation from servers through a custom protocol. Communication works through a signal channel, with an optional data channel available for provisioning of clients.	Fully automatic mitigation is possible, even across multiple domains by using recursive signaling. While signaling for help, all relevant information about the attack are provided by the client and can be distributed among all IETF DOTS instances. Provisioning clients through the data channel allows to share configuration as well as white- and blacklists with all clients.	The signal as well as data channels used for communication can be congested by the attack. The client server model makes a highly distributed defense network more complicated to set up and maintain.
	CoFence	CoFence relays attack traffic to mitigation instances to filter out malicious packets and return filtered traffic back to the system under attack.	Circumventing the resource-intensive packet inspection by relaying the traffic to be cleaned is an efficient defense approach. The use of NFV for deployment makes CoFence simple to install and maintain.	Collaboration among CoFence instances is limited to relaying traffic. High bandwidth attacks could overwhelm the CoFence instance, rendering it unable to relay traffic for cleaning.
	Bohatei	Bohatei demonstrates the scalability and performance advantages of using SDN in conjunction with NFV and building a DDoS defense system on top of proven, existing mitigation components.	Leveraging NFV in conjunction with SDN makes Bohatei up to 5.4 times more cost effective compared to fixed defense facilities. Utilizing existing mitigation solutions like Snort, Bro and iptables makes sure, proven defense technologies are employed.	The implementation by Fayaz *et al.* does not include cooperative aspects of the mitigation.
	DefCOM	DefCOM uses an overlay network to facilitate better coordination among peers with inherent scalability. Attack packets are dropped by peers located at the attack source.	The P2P overlay network is inherently scalable and easy to set up. Only a small number of pre-defined messages are required for communication. Traffic is routed entirely on the underlying routing protocols, keeping the overlay network open for mitigation purposes. Source-based mitigation in a cooperative manner makes attack detection easier.	The overlay network still relies on the underlying data links and could be congested. Propagation of messages in the overlay network might take a long time based on the size of the network.

Table 2.1 provides a complete overview of all DDoS defense systems discussed in Section 2.1 and Section 2.2. This overview should serve as a recap of each system to better understand the overall divide between centralized and decentralized defense mechanisms as well as the multitude of communication schemes employed by the systems.

Chapter 3

Approaches Toward a Proof-of-Mitigation

One of the remaining challenges not addressed in any of the presented decentralized, cooperative DDoS mitigation systems in Section 2.2 is an effective incentives scheme to bolster cooperation among all parties involved in the collaborative defense effort.

For ISPs in context of the BloSS system, cooperative relations might consist of Service Level Agreements (SLAs) to establish a well-defined trust on which a mitigation service can be built [58]. In a more general setting with only a few participants in a distributed DDoS mitigation scenario, setting up SLAs for the purpose of provisioning the required filtering rules would be a valid and efficient solution, however considering a global-scale mitigation network, the additional legal overhead created by SLAs might make such a system too costly to deploy. This is where the aforementioned incentive scheme can be leveraged: By substituting the need for trust based on legal frameworks such as SLAs with a payment-driven incentive scheme backed up by transparency and security measures, the scalability of the entire cooperative defense system is largely improved.

To enable transparent provisioning of mitigation-related network filter rules, a replicable, deterministic format needs to be chosen for the filters to be deployed. Furthermore, the successful deployment of the filters and subsequent completion of the mitigation service needs to be backed up by a Proof-of-Mitigation that can be communicated directly to the service user.

The research part of this thesis presents multiple approaches toward such a Proof-of-Mitigation that allows to automatically verify the successful completion of a mitigation request and therefore enables an efficient incentive scheme with fully automated payouts. The detailed descriptions as well as pros and cons in regard to each approach are later followed by a qualitative comparison of all approaches to discuss similarities between them and to determine which approach might be best suited to deliver the desired mitigation proof.

The metrics upon which the qualitative comparison in Section 3.5 is based, include the following:

1. **Confidentiality**: Describes how effective rules and measures are to protect both the mitigation proof as well as the mitigation system from unauthorized access.

2. **Integrity**: Defines the level of trustworthiness of the proof generated by the given approach in regards to accuracy and tamper-resistance.

3. **Availability**: Relates to the availability of the mitigation proof to authorized parties.

4. **Reproducibility**: Describes the ability to reproduce the proof through replay by a third party.

5. **Tamper-Evidence**: Discloses the effort necessary for changing the proof of mitigation to reflect an alternate reality.

6. **Timeliness**: Defines the ability to provide an automatically verifiable proof within a pre-specified time-frame while adhering to all security requirements.

7. **Deployment Complexity**: Defines the additional resources required to deploy the approach.

8. **Scalability**: Relates to the adaptability of the approach in regards to a large-scale DDoS defense scenario with high-bandwidth attacks.

9. **Service Model**: Defines the cloud service model the approach most closely relates to.

All approaches presented in this chapter have been accepted for the 2018 IEEE CyberSciTech conference in Athens compiled as a work in progress paper. The full paper is included in the thesis as Appendix A.

3.1 Marketplace of VNFs for Mitigation

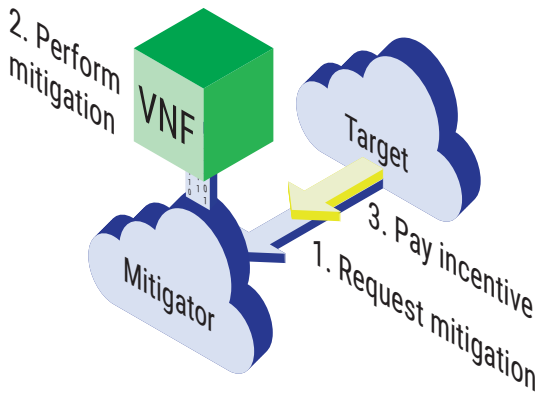

Figure 3.1: Mitigation service based on VNFs available in a trusted marketplace

Deploying a complex mitigation system such as the BloSS on traditional networking equipment generates high CAPital EXpenditures (CAPEX) to set up and maintain the dedicated networking hardware as well as OPerating EXpenditures (OPEX) to provide a mitigation service on such an infrastructure.

To minimize both CAPEX and OPEX, virtualized networking equipment, realized through Network Function Virtualization (NFV)[32], can be leveraged to quickly deploy the mitigation system and run it directly on commodity hardware. This allows the operator of an AS interested in providing a DDoS mitigation service to their customers to quickly instantiate such a service and run it as long as interest in the service persists, with the option of completely removing the entire service without any change in the underlying networking infrastructure.

The isolation provided by NFV is not only a good solution to deploy the BloSS on a large scale but can also serve as a cornerstone for a mitigation proof. Running the whole system inside a Virtual Machine (VM), as is the case with VNFs, we are presented with a closed-off environment that allows us a high degree of control and therefore reduces the number of potential variables to account for when trying to proof a successful mitigation.

Distribution of these isolated mitigation instances could be accomplished by utilizing a marketplace concept: Each AS operator interested in setting up the mitigation service could directly procure the VNF from the marketplace and instantiate inside their own infrastructure. The marketplace would also ensure transparency of the provided VNFs since everyone can examine them on the publicly accessible marketplace and hashed checksums of the VNFs could be used to ensure the integrity of the VNF image. An additional component of the marketplace that could be used to enhance the validity of the mitigation proof is the logging capability of such a marketplace: Every time an AS operator requests the VNF from the marketplace, a log of the activity can be kept on the marketplace, allowing target AS operators to verify whether a mitigation service provider actually obtained the mitigation VNF from the marketplace to provide the service in the first place.

Infrastructure requirements to run a VNF-based DDoS mitigation system only include data plane control and can even be met without a Software-Defined Networking (SDN) platform present [32].

This approach leverages implicit trust through the isolation provided by the virtualization. The burden of proof in regards to the successful completion of the mitigation service does however still lie with the mitigation AS, which means they have to unequivocally proof that the mitigation VNF running inside their networking infrastructure has not been compromised — neither by them, nor by a third party. This burden is hard to overcome, especially since the use of commodity hardware in data centers running VNFs does traditionally not provide a way to establish trust in its integrity. Even if the integrity of the VNF could be guaranteed, the underlying network flows could be rerouted to circumvent the filtering and resulting use of computing resources. This could be orchestrated by a malevolent mitigator to still receive the automatic incentive payout without providing the actual service, which is called free-riding [10] and needs to be prevented to establish a sustainable incentive scheme.

3.2 Trusted Computing

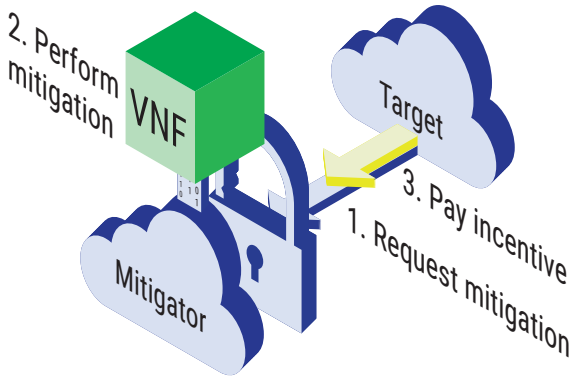

Figure 3.2: Mitigation service based on Trusted Platform Modules (TPM) and Intel Software Guard Extension (SGX)

As outlined in Section 3.1, commodity-based networking infrastructures are susceptible to outside influences such as hypervisor introspection, where the VNFs state can be viewed and modified at will by the operator running the hypervisor [51]. The problem that this lack of integrity and confidentiality presents in regard to a mitigation proof is the need to convey successful service completion by guaranteeing that the service has not been tampered with during operation.

Utilizing trusted computing and specifically the Trusted Platform Module (TPM) has been demonstrated in [35] to be a viable way of establishing a base trust in the system: The TPM module allows the secure storage of verification hashes that can be used to attest a secure boot chain from BIOS over bootloader, operating system (OS) up to the hypervisor. To include the VNF itself in the chain of trust, approaches like ”vTPM” [41] provide a virtualized TPM that aims to ensure the integrity of the VNF itself. Virtualized trusted computing does however lack the base hardware trust available through TPM, which is why vTPM should be used with an additional remote attestation component to have the integrity of the VNF image repeatedly checked by a remote third party. Traditionally, remote attestation is often only used for platform integrity checks. To that end, Ravidas *et al.* [45] propose a remote attestation server that allows attestation of VNFs through image integrity checks. This is accomplished through an external Trusted Security Orchestrator (TSecO) [45] which receives VNF launch requests from a Virtual Infrastructure Manager (VIM) running inside the ASes networking infrastructure. The TSecO subsequently decides to allow or deny the launch request based on a whitelist holding checksums for allowed VNFs. Such a whitelist could be deployed on a VNF marketplace as outlined in Section 3.1 for easier distribution and accountability.

To address scalability concerns in regards to the centralized aspect of a remote attestation system, a decentralized approach could consist of known good values stored in a distributed data structure. All parties would have access to that data store. The decentralized attestation would work similar to the way the work of Ruan *et al.* [47] approaches the problem: Attestation for VNF integrity of an AS M in the role of the mitigator would be performed by the target AS T, since its interests directly correlate with the correct

execution of the mitigation code running on AS M. Access to the mitigation VNF would be required by AS T to check the image integrity by comparing it to the decentrally stored known good values.

The chain of trust only guarantees the integrity of the system and the VNF images before they are instantiated [1]. To ensure the integrity of the VNF during runtime and especially to protect the data inside the VNF, partial isolation through technologies such as Intel Software Guard Extension (SGX)[6] which provides a secure enclave to store application states as proposed by Shih *et al.* [51], can be used. With this secure storage element in place, the chain of trust includes all critical components of the mitigation system.

The biggest drawback of using a trusted computing approach to guarantee the integrity of the mitigation system are the strict hardware requirements: The TPM, as a discrete chip or integrated solution, does not come standard on all server motherboards which restricts the choice of commodity hardware for the NFV-enabled networking infrastructure of the mitigation AS. The same is true for Intel SGX, which would limit the choice of processor to Intel's range of products, getting more cost-effective competitors like Advanced Micro Devices (AMD) left out.

The burden of proving the mitigation still clearly lies with the mitigator, similar to the approach discussed in Section 3.1, however in the trusted computing scenario, the operator of the mitigation system does also give up freedom to run arbitrary VNFs on their hardware since only signed VNFs approved by the TPM can be instantiated. With the addition of remote attestation, only VNFs featured on the whitelist distributed through the VNF marketplace can be instantiated, which limits the level of freedom for the mitigator to run arbitrary code even further.

Despite all efforts to ensure the integrity of the entire system, the trusted computing based approach does still suffer the problem described in Section 3.1 in regards to the integrity of the underlying network flows, since these cannot be isolated into the chain of trust.

3.3 Secure logging

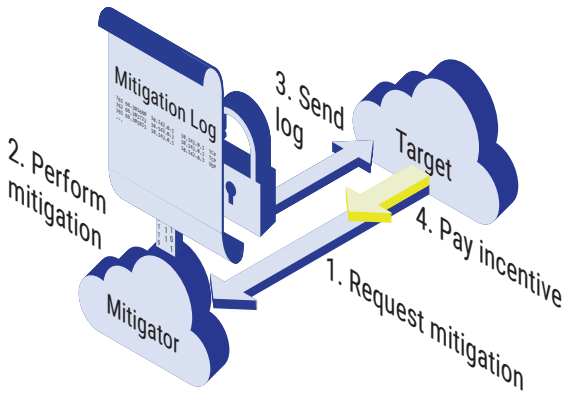

Figure 3.3: A mitigation log is provided to the target AS

The importance of providing accountability for the network flows and whether they have been handled in their entirety without rerouting parts of them to reduce the amount of computing resources necessary to provide the mitigation service has been shown to be one of the drawbacks of both the NFV-based approach in Section 3.1 as well as the trusted computing approach in Section 3.2. Both approaches discussed so far focused on the mitigator to provide a proof of successful service completion by ensuring the integrity of the system the service is running on. Since the AS operator in the role of the mitigator has full control over their networking infrastructure to apply the required filters, which allow blackholing [30] the attack traffic indicated by the attack target and service user, the task of providing the Proof-of-Mitigation naturally falls upon them.

Since all of the traffic passing through the mitigation system can however also be recorded through a mitigation log, which is oftentimes an inherent component of a mitigation system, a tangible product, the mitigation log, exists, that can be used as a proof of mitigation. By transferring this log from the mitigator to the service user, independent verification of service completion can occur and at that point, the mitigator does not need to be trusted anymore.

To make sure that the log presents a tamper-proof medium to convey the successful filtering of traffic, a scheme similar to the one presented in [50] by Schneier *et al.* can be used. In this scheme, the entire log as well as individual log entries are secured by an authentication key [50]. Furthermore, a hash-chain is established which makes the entire log tamper evident since every change of an existing log entry would break the chain and clearly show the anomaly reflected in the incorrect hashes produced by the altered log [50]. Security schemes like these have already been successfully applied to practical applications such as the work presented by Nguyen *et al.* [33] where a cloud-based application enabled the storage of "secure, time-synchronized and tamper-evident logs"[33] received from multiple medical devices via a dongle storing the actual data. By leveraging technologies such as Intel SGX and TPM in addition to the cryptographic scheme devised by Schneier *et al.* [50], integrity and confidentiality of the data can also be ensured since the logs can be stored securely on the system protected by the hardware chain of trust [33]. This is not only a good fit for the sensitive medical data handled by Nguyen *et al.* but could also be utilized for our Proof-of-Mitigation problem since it would allows us to ensure the integrity of the log, which could potentially be comprised by the mitigator since they have full control over their infrastructure and could replace the entire log with a falsified version without providing the actual mitigation service. Intel SGX has also been proven to be a good fit to ensure NFV integrity as discussed by Shih *et al.* [51] since it allows the secure storage of application states inside the secure enclave which could directly be used as a storage location for the mitigation log while it is being compiled by the mitigation VNF.

After the log has been created by the mitigation system, the verification stage can be designed similar to a remote attestation approach as discussed in Section 3.2. An attestation scheme specifically geared towards log files has been presented by Haeberlen *et al.* [13] in the form of a remote audit system. Here, a focus lies on deterministic log files that are checked by replaying system executions and comparing the resulting log to the one under test to determine the correct behavior of the mitigation system.

The complexity reduction of creating the tangible mitigation proof in the form of a log file lifts the burden of proving successful mitigation service completion from the mitigator but does also come with some drawbacks. One of them is the size of these logs, which can become quite large since they try to reflect mitigation endeavors against high-bandwidth DDoS attacks. The file size is especially problematic when considering the need to transfer the entire log to the remote audit system for verification. This clogs the vital management networks used for mitigation signaling and introduces additional delays based on transfer times of the log file.

The problems do however already occur at the mitigation stage, since the log files need to be stored securely to ensure integrity and possibly confidentiality, hardware trust through Intel SGX or a TPM is required, which results in the same limited choices of potential hardware to be used for the networking infrastructure as outlined in Section 3.2.

Finally, the remote audit procedure entirely relies on deterministic log files [13]. However, this is not the case for log files created through networking events since replaying the traffic for large-scale DDoS attacks to check the resulting network logs is infeasible due to the lack of knowledge about the overall traffic after it has occurred.

3.4 Network Slicing

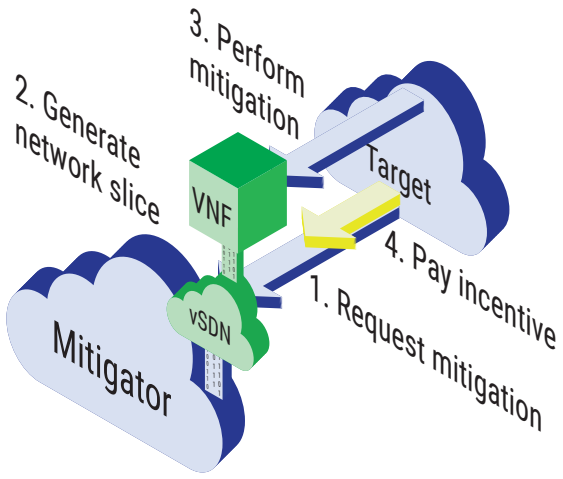

Figure 3.4: Virtual network slices are provided to run the mitigation service

In the same vein as the secure logging approach discussed in Section 3.3, network slicing shifts the responsibility of providing the proof of mitigation to the service user itself. This is possible, because of recent advances in network virtualization technologies and specifically Software-Defined Networks (SDN), that allow to tailor a networking infrastructure to specific use cases. In the context of a mitigation service, network slicing is the most intriguing part of SDNs, since it allows to carve a network up into separate parts that can be used independently with each slice being configured with a certain set of network flows [59].

The work by Zhou *et al.* [59] demonstrates how network slicing can be automated to build a "Network-Slicing-as-a-Service (NSaaS)" [59] which can be used for mitigation purposes by automatically configuring a network slice with the network flows of interest based on the attacking IP addresses provided in an attack report sent by the target AS. Limiting access only to these network flows allows clear isolation of the service user from the rest of the networking infrastructure and allows them to directly run the mitigation remotely on the provided network slice running inside the mitigator AS networking infrastructure.

The NSaaS solution discussed by Zhou *et al.* only considers configuring existing network slices but the entire process can be automated even further, for example by utilizing automated slice generation. Bozakov *et al.* propose "AutoSlice"[3] which is able to create on-demand virtualized SDN networks (vSDN)[3]. The mitigation service user could now be provided with access to a VNF running the mitigation service in the form of an SDN controller to automatically apply the filters according to the IP addresses provided by the user.

This automatic generation and configuration of network slices is beneficial for the overall incentive scheme since the service user now has full control over the entire mitigation and payout for successful completion can automatically be triggered after the filters have been applied by the service user on the provided vSDN-based network slice. The high degree of automation thanks to the NSaaS[59] and "AutoSlice"[3] approaches allows to eliminate delays in the mitigation since a service request by the target AS would directly instantiate the vSDN as well as the mitigation service VNF with full access to the vSDN to apply the filtering in order to stop the attack.

The shift of responsibility in regards to proving the successful completion of the mitigation service from the mitigator to the target does however not fully eliminate the requirement to trust that the mitigator provides full access to the required network flows and is not rerouting any of the attack traffic in order to facilitate free-riding for themselves.

From an implementation perspective, network slicing presents some strong hardware as well as software requirements to fully realize the automated slice generation and configuration. The entire networking infrastructure needs to be SDN based to allow network slicing in the first place and additional Open vSwitch installations are required to circumvent the limited flow-table sizes present in most OpenFlow hardware switches [3].

3.5 Discussion

From Table 3.1, we can see that no single approach achieves the required balance between practicability and security.

It is important to note that throughout the qualitative comparison, low generally means bad and high means good, however in regards to the deployment complexity, the situation is reversed, where a low deployment complexity is good, whereas a high deployment complexity is bad.

Table 3.1: Qualitative comparison of approaches to provide a Proof-of-Mitigation

	NFV	Trusted Computing	Secure Logging	Network Slicing
Security				
1. Confidentiality	Low	Medium	Low	Low
2. Integrity	Low	High	High	Low
3. Availability	High	Medium	High	Medium
4. Reproducibility	High	High	Low	High
5. Tamper-Evidence	Low	Medium	Medium	Low
6. Timeliness	High	High	Low	Medium
Practicability				
7. Deployment Complexity	Low	High	High	High
8. Scalability	High	Low	Low	Low
Scope				
9. Service Model	SaaS	SaaS	PaaS	IaaS

3.5.1 NFV vs. Network Slicing

There exists some overlap between the different approaches due to similar technologies used. This can be seen between the NFV and Network Slicing approaches, which exhibit similar security characteristics with the NFV approach slightly edging out the Network Slicing approach due to the additional overhead created by the slice creation of the latter as discussed in Section 3.4.

The clear differentiator between the NFV and Network Slicing approach are the practicability aspects: While the NFV approach can be deployed quickly and efficiently, even on networking infrastructures which are not SDN-based, the network slicing approach produces additional overhead for each slice creation and requires SDN-based networking together with additional Open vSwitch servers to circumvent limited flow-tables sizes and allow the "AutoSlice"[3] approach to work.

The NFV approach also has the edge when considering scalability of both approaches: While VNFs can be quickly spun up on demand across multiple ASes, Network Slicing is limited by the additional hardware requirements and higher resource consumptions to enable automatic slice generation and configuration and can therefore potentially not be scaled out globally since only a limited number of ASes meet the requirements to enable network slicing.

3.5.2 Trusted Computing vs. Secure Logging

The overlap between the trusted computing and secure logging approaches mainly concerns the use of technologies like TPM and Intel SGX by both approaches. While the Trusted Computing approaches core idea is based on utilizing these technologies to reduce the amount of trust necessary toward the mitigator, the core idea of the Secure Logging approach, namely keeping a mitigation log to serve as a tangible Proof-of-Mitigation, is

already an inherent part of the mitigation process. Trusted computing aspects are only added to the Secure Logging approach to ensure integrity of the log, which cannot be provided through the existing logging behavior.

Utilizing both TPM as well as Intel SGX makes both approaches well suited to serve as a Proof-of-Mitigation in respect to the security requirements, however they both exhibit a high deployment complexity and low scalability due to the strong hardware requirements which goes to show once again that finding the balance between security and practicability is a very important aspect to consider in regards to a true Proof-of-Mitigation.

3.5.3 Combining Approaches

We already established that no individual approach is capable of providing a comprehensive mitigation proof since none of them are able to strike a satisfiable balance between security and practicability. It may therefore appear that combining some of these approaches could lead to a comprehensive solution, however as noted in Section 3.5.1, since there exists overlap between individual approaches, a combination of them would multiply the disadvantages by combining disadvantages of both approaches while merging the overlapping advantages to produce a combination approach that would fare even worse in regards to the balance between security and practicability.

3.5.4 Cloud Service Models

Instead of comparing the individual approaches in regards to their overlapping technologies, the cloud service model metric represented as characteristic number 9 in Table 3.1 helps to differentiate the approaches by correlating them with their respective cloud service model. NFV and Trusted Computing represent the same model of Software-as-a-Service (SaaS)[24] since they both utilize NFV-based mitigation systems which are available on demand to the service user while Secure Logging follows a Platform-as-a-Service (PaaS)[24] model since it provides an interface to the mitigation service which in turn produces the mitigation log as a product. Network Slicing allows the highest degree of access to the mitigation service, which clearly makes it an Infrastructure-as-a-Service (IaaS)[24] model where the on-demand vSDNs represent the infrastructure being provided.

Chapter 4

Design

This chapter details design decisions that went into the re-engineering of the Blockchain Signaling System (BloSS). The re-design of BloSS is based on the work by Rodrigues *et al.* [46]. They demonstrated that a multi-domain cooperative DDoS defense system can be realized by utilizing the blockchain as a communication medium and leveraging the recent advances in SDN to simplify the deployment of the entire system.

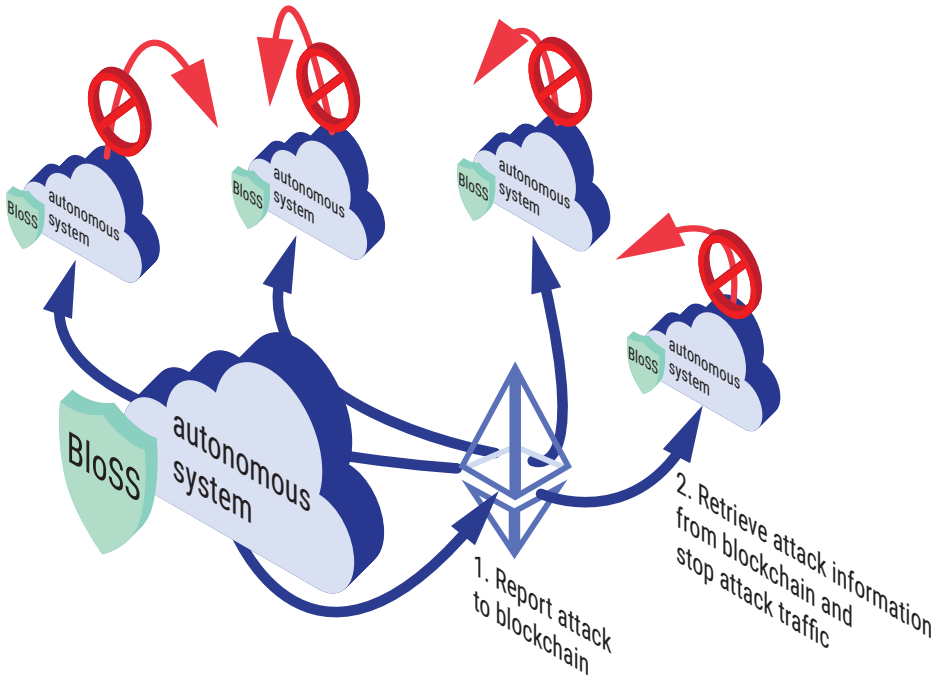

Figure 4.1: Multi-domain DDoS defense through the BloSS

The BloSS is a decentralized DDoS defense system as illustrated in Figure 4.1 where each AS taking part in the multi-domain, cooperative defense alliance and running the BloSS is able to post information about an ongoing attack to the Ethereum blockchain[57]. The attack information on the blockchain can then directly be accessed by all other ASes to block the attack at the source and mitigate the entire threat.

The re-design of the BloSS builds upon the existing design ideas of Rodrigues *et al.* and mainly strives to modularize the existing monolithic architecture as outlined in Section 1.2. The goal of the modularization is to decouple the main defense system logic from the underlying network infrastructure by delegating all network-specific tasks to a separate module. To further increase loose coupling between the individual parts of the defense system, all data exchange related tasks apart from networking are also separated into a purpose-built module.

This high degree of modularization aims to allow the adaptability of the entire system to different computing environments, including networking infrastructures apart from the SDN-based networking focused on in the original implementation by Rodrigues *et al.* [46]. In addition to that, modularizing the data exchange capabilities of the BloSS allows to switch to a different blockchain or data store in the future.

4.1 Architecture

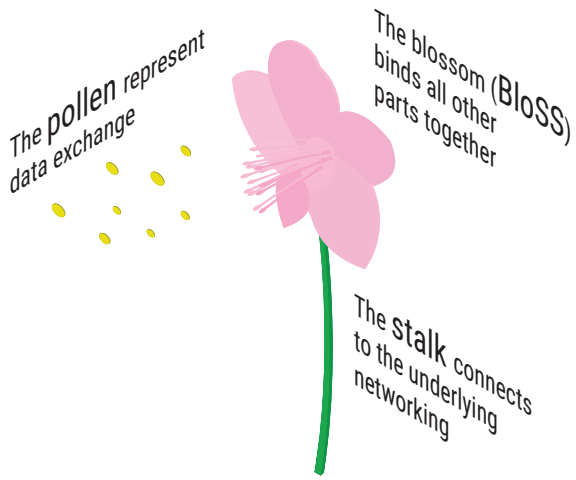

Figure 4.2: Metaphor for the naming scheme of the individual BloSS modules

To better grasp the tasks of the individual modules, a metaphor was used as a basis for the naming scheme. Figure 4.2 shows a flower, with each part of the flower representing a vital module of the BloSS. The blossom of the flower represents the core module of the BloSS which uses all other modules to mitigate an attack. Data exchange is accomplished with the ”Pollen” set of modules and network-related tasks are handled by the ”Stalk” module. Pollen is divided into dedicated modules for the individual data exchange duties of the BloSS. This includes a blockchain module for access to the Ethereum blockchain, a data store module managing data on the Inter Planetary File System (IPFS)[2] as well as a database module to store statistics on InfluxDB for demonstration purposes.

An overview of the entire re-engineered BloSS architecture is provided in Figure 4.3 detailing the connections between all modules. The separation of BloSS and Stalk is contrived of a REST interface [17] to facilitate the isolation of the BloSS module, possibly encapsulating the entire module together with Pollen blockchain and Pollen datastore inside a

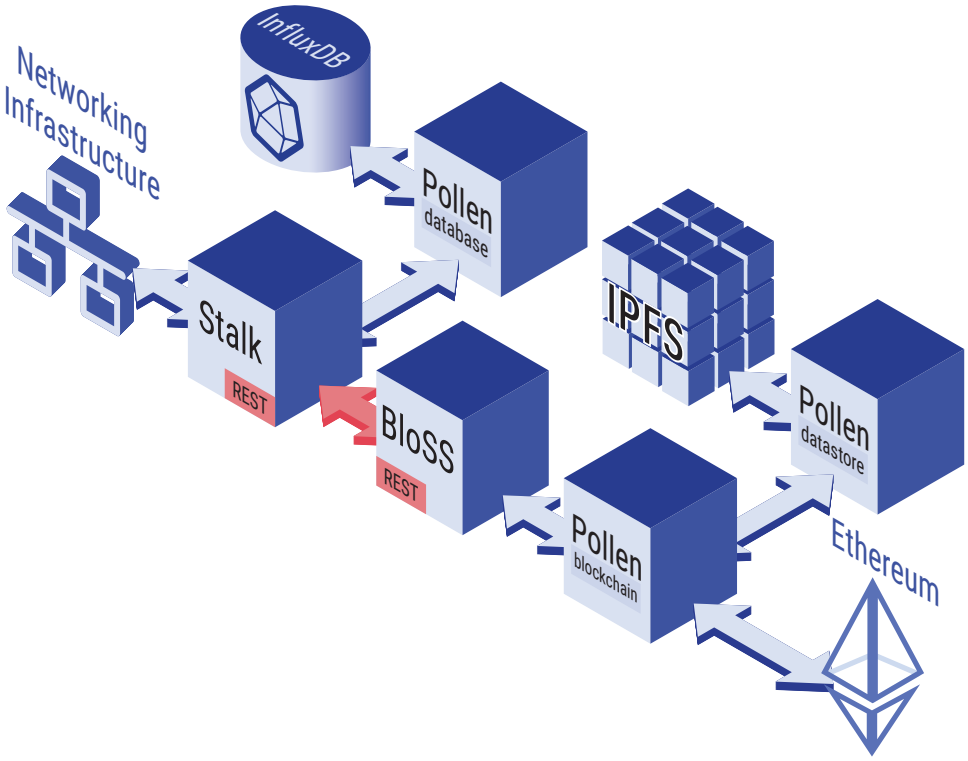

Figure 4.3: Architecture of the Blockchain Signaling System (BloSS)

VNF. This design decision was made with findings from Chapter 2 in mind to keep the architecture simple enough for a potential implementation of a Proof-of-Mitigation.

Attack information posted to the blockchain is not directly stored on the blockchain due to limited block sizes. For this purpose, IPFS is used as a decentralized and highly scalable storage solution to hold attack information. Each AS running the BloSS also maintains an IPFS node to enable the decentralized storage. Whenever a new set of attack information is posted to the blockchain, the data is first stored in IPFS and only the hash as a unique identifier of the storage location within IPFS, is stored in a block on the Ethereum blockchain.

4.2 Security Considerations

Pollen datastore also includes an encryption component which is not directly visible in Figure 4.3 since it is an inherent part of the entire module. The encryption of attack information posted to IPFS ensures the confidentiality as well as the integrity of the attack information based on a per-message signature bundled with the attack information. Confidentiality is an important attribute of the data exchange between ASes since the attack information can be sensitive in regards to implicating individuals both as victims of an ongoing DDoS attack or as the perpetrators of said attack.

Verifying the integrity of the attack information allows to hold each AS accountable for the information posted to the blockchain and makes forgery of attack information impossible.

The integrity-check is enabled through a public key published by each AS to the blockchain and therefore available to all ASes participating in the BloSS defense alliance. Without this measure in place, forgery of attack information could allow a malevolent party to indicate specific IP addresses as being the source of an ongoing attack and in effect blocking flows from these addresses to the target address specified in the attack information.

In addition to encrypting each set of attack information when posted to IPFS, all communication between the Pollen datastore module and IPFS is encrypted with the libp2p-secio[21] stream security transport which is based on TLS 1.2. Transport encryption would not be strictly necessary since the data being transported is already encrypted, however this allows a certain degree of anonymization for the defense system users since it is not possible to ascertain which AS accessed which attack information when simply looking at the communication between IPFS and AS. This added anonymity can also be seen as an additional factor contributing towards increased confidentiality.

Communication between individual Ethereum nodes is also encrypted as detailed in the DEVp2p white paper by Gavin Wood [56], which could again contribute to increased confidentiality, however it is important to note that due to the distributed ledger characteristic of Ethereum, all transactions that change the blockchain can be traced back to the party responsible for the chain which means anonymity is not provided for anybody able to access the blockchain.

The last part of the communication chain with the REST interface between BloSS and Stalk is not encrypted. This is by design since the REST interface is designed to only be accessible on the same machine to allow for simple communication between BloSS and Stalk while enabling a high degree of encapsulation for the BloSS module in order to allow the implementation of Proof-of-Mitigation schemes as discussed in Chapter 3.

4.3 Defense Scenario

Figure 4.4 shows a prototypical defense scenario involving a mitigator as well as target AS. Attack detection is outside the scope of the BloSS so the first step includes compiling the attack information and encrypting it to later store to IPFS and post the IPFS hash to the Ethereum blockchain.

To minimize access to IPFS as well as Ethereum to access attack information and the public key of the target AS, the attack information hash is connected to a boolean indicating whether the information has already been accessed by the mitigator AS in order to block the attackers.

Incentive schemes necessary to realize a true Mitigation-as-a-Service (MaaS) offering as outlined in [23] are out of the scope of this implementation of the BloSS. This is mainly due to the lack of availability of a satisfactory Proof-of-Mitigation approach as outlined in Chapter 3 but also based on the demonstration and research nature of the implementation which aims to provide a flexible platform to test concepts in order to pave the way toward a true MaaS offering.

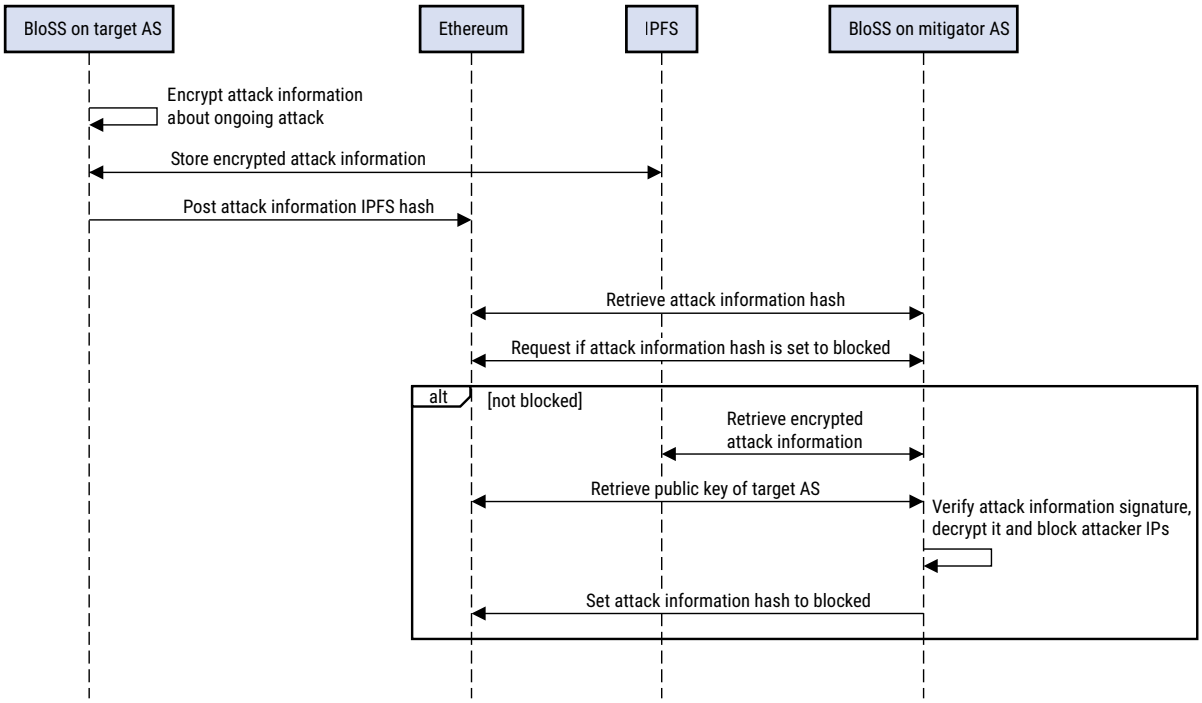

Figure 4.4: BloSS defense scenario including a target and mitigator AS

4.4 Attack Information

The attack information is the communication payload exchanged between individual BloSS instances. It carries the relevant data to indicate the target as well as the source of an attack, which enables the collaborative mitigation of a large-scale DDoS attack directly at the source. Following data points are part of the attack information:

- **Target**: The IP address being targeted by a DDoS attack
- **Action**: Action to take against the attack. In the version of the BloSS outlined in this thesis, this is limited to the ”blackhole”[30] action discarding attack traffic directly.
- **Timestamp**: The purpose of the timestamp is to make sure that outdated attack information is no longer considered for blocking, making sure that no unwanted side-effects can occur.
- **Subnetwork**: The subnetwork, which all attacker addresses are part of
- **Addresses**: IP addresses of the attackers
- **Hash**: A hash computed from the target address, timestamp, subnetwork and action. This hash is used to uniquely identify a set of attack information.

A unique set of attack information will henceforth be called an ”attack report” for the remainder of the thesis. A single attack might often produce multiple attack reports since

each report represents a specific subnetwork of attackers to simplify the mitigation of the attack. Based on the subnetwork, the attack report can directly be sent to the AS managing the subnetwork from which the attack originates.

Chapter 5

Implementation

This chapter details the technical implementation of the re-engineered BloSS. Implementation details are provided top-down beginning with the demonstration system on which the BloSS was deployed for development and evaluation purposes and going down to the individual components of each module of the entire system.

5.1 Demonstration System

The development and demonstration system for the BloSS is realized as a physical single board computer cluster as illustrated in Figure 5.1, complete with the necessary networking equipment to allow the individual computers to talk among each other. Most parts of the system already existed when I started my thesis since it is part of earlier work by Rodrigues *et al.* [46] to demonstrate the viability of using the blockchain as a signaling strategy in a multi-domain DDoS defense system. Throughout the course of my thesis, two additional single board computers were added to serve as auxiliary controllers as illustrated in Figure 5.1.

It might seem odd that the choice for such a demonstration system fell on physical hardware instead of leveraging virtualization technologies like Mininet[52] that would allow the entire system to be reproduced on a single, high-powered server. The rationale behind this decision is that the physical form makes the whole cooperative defense concept more tangible and facilitates straight-forward demonstration of the system. In addition to that, it allows to observe realistic networking congestion based on hardware- as well as software-related shortcomings instead of relying on virtualized networking hardware that might behave more predictable but less realistic.

5.1.1 Networking

Two main networks are set up in the demonstration system: A general network, used to route the DDoS attack traffic and a management network through WiFi, that allows the

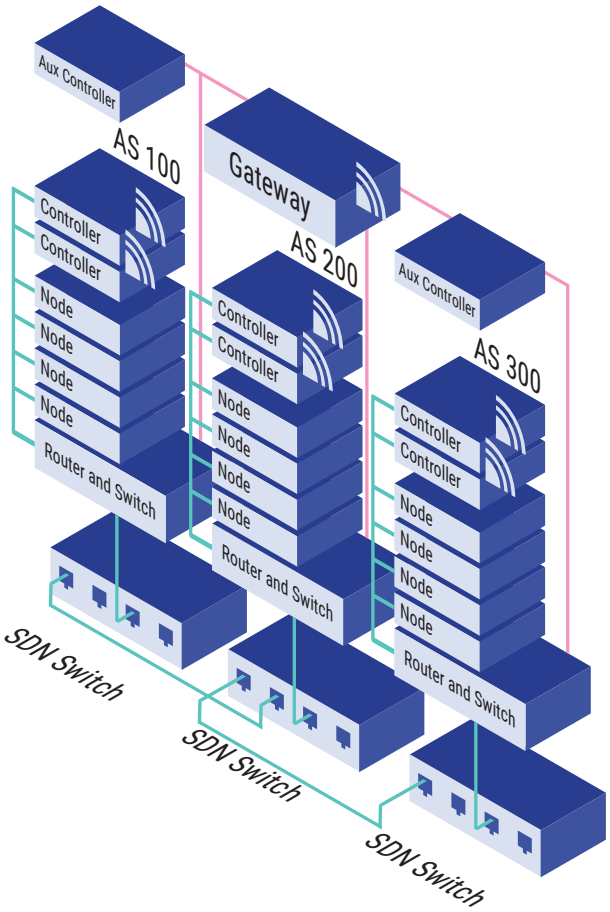

Figure 5.1: The BloSS demonstration system

BloSS instances to communicate with each other on an uncongested channel. Figure 5.1 shows a simplified model of the system with an emphasis on the general network to carry the attack traffic. The management network is depicted with the wireless signal symbols as well as the pink connections between the three routers, the two auxiliary controllers and the gateway.

The gateway as the fourth routers in the system interconnects the other three routers and provides the access point for the WiFi connection which the management network relies on. Apart from providing the maintenance network, the gateway allows the developer to easily access all computers in the system.

The system is split up into three separate ASes: AS 100, AS 200 and AS 300 with each AS consisting of four compute nodes used to initiate the attack traffic and two controllers, which hosts the BloSS as well as the Ethereum and IPFS nodes. The controllers are connected both to the wired general network as well as wirelessly to the management network. In addition to the per-AS controllers, two auxiliary controllers host system-wide a Ethereum Netstats[36] dashboard as well as an InfluxDB[15] installation together with the Grafana dashboard[20]. These auxiliary services are solely for development and demonstration purposes and are not vital to the overall BloSS.

The SDN part of the system is driven by three Zodiac FX SDN switches from Northbound

Networks [31] which are interconnected with each other and in turn to four compute nodes and one controller in their respective AS, which are connected to a small router and a switch. These additional routers and switches are necessary since the Zodiac FX switches only provide four ports, which is not enough to connect all hosts of an AS. The SDN controller responsible for the Zodiac FX switches is directly specified in the Zodiac FX web interface. For the demonstration system, the three lower controllers in the setup have the role of SDN controllers.

5.1.2 Single Board Computers

As briefly mentioned in Section 5.1.1, the single board computers in the demonstration system are divided into compute nodes to launch DDoS attacks, controller nodes running the BloSS and auxiliary controller nodes providing statistical services for development and demonstration purposes.

The compute nodes as well as the controllers are ASUS Tinker Board single board computers with a 1.8 GHz quad core ARM Cortex A17 CPU and 2 GB of RAM. The auxiliary controllers are Pine64 Rock64 single board computers with 1.5 GHz quad core ARM Cortex A53 CPUs and 4 GB of RAM. The Tinker Boards are equipped with slower microSD-based storage, whereas the Rock64 boards feature eMMC storage which provides better read and write performance, specifically geared toward the InfluxDB running on one of the auxiliary controllers.

5.2 Ethereum Blockchain

Arguably the most important part of the BloSS is the Ethereum blockchain used as a communication channel between ASes. The variant of Ethereum that was used for the re-engineered BloSS is based on the Proof-of-Authority (PoA)[29] consensus mechanism.

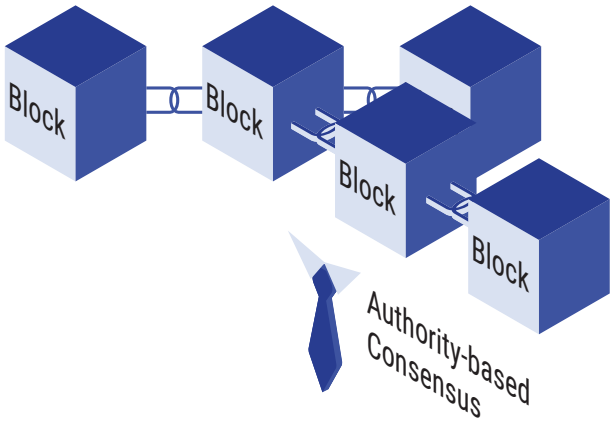

Figure 5.2: The Proof-of-Authority consensus mechanism

Instead of relying on computationally complex cryptographic puzzle to form a consensus as it is the case with the Proof-of-Work (PoW) mechanism, the PoA mechanism appoints

some of the Ethereum nodes as signers with the power to find consensus in the blockchain and therefore to define which blocks are part of the chain and which are not. Figure 5.2 depicts a situation, where a fork in the chain lead to a consensus decision to go with the lower fork as the prevalent chain.

The advantages of using a PoA-based blockchain are reduced computing power requirements in comparison to PoW and the ability to define an arbitrary block period. Adjusting the block period freely allows faster development and debugging of the system since very short periods of only a few seconds can defined. Designating specific Ethereum nodes as the signers of the blockchain could be based on the size of the AS in the case of the BloSS. Larger ASes are less likely to vanish from one day to the other and they have more reputation to loose for abusing their blockchain signing rights.

The demonstration system uses a private PoA-based Ethereum blockchain [49] with a block period of 5 seconds. This period has been chosen since each block consumes around 1024 bytes of storage space [49] which could quickly lead to a lot of wasted space if the block period is set too low.

5.2.1 Signaling Attacks

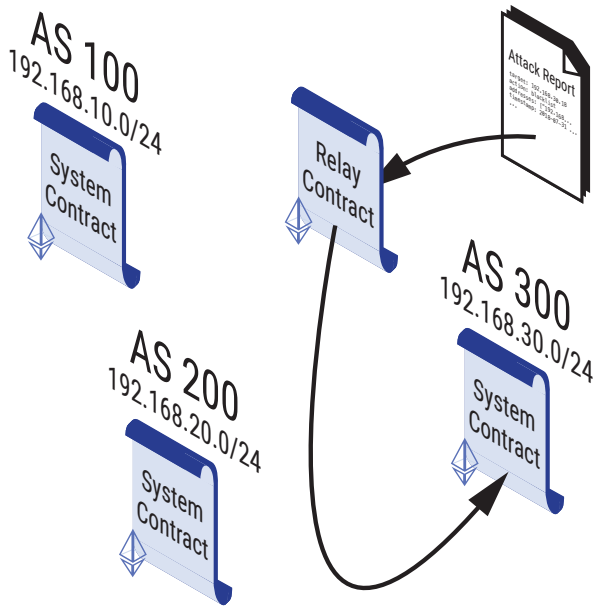

Figure 5.3: Ethereum contract setup with the relay and system contracts

We already saw in Section 4.4 that the attack report specifies the subnetwork of the attackers. This is used to enable efficient signaling among a large number of ASes in the Ethereum blockchain. A single relay contract keeps track of all the subnetworks participating in the collaborative DDoS defense and maps each subnetwork to the system contract managing that subnetwork. As soon as an attack report is posted to the relay contract, it directly sends it to the correct system contract to allow the responsible BloSS instance to act on it.

This relaying of attack reports represents a significant change from the old system, where only a single smart contract was responsible for all attack reports. With the new system, it becomes possible to implement a simple access control scheme where only the owner of a specific system contract is able to access the attack reports stored in that contract. Due to the openness of the Ethereum blockchain, this simple scheme does of course not deter determined third parties to access the attack reports stored in any contract by examining the individual blocks and finding the point at which the report has been stored. Apart from simple access control, this scheme does however also allow to reduce the number of calls necessary for a specific BloSS instance to get the attack reports it is interested in, *e.g.* the ones mentioning attackers from a subnetwork it is managing.

The number of subnetworks managed by a single system contract is not limited and the system contract simply registers the subnetwork for which it wants to receive attack reports with the relay contract.

System contracts also represent an important part of the encryption scheme since the public keys required to encrypt symmetric keys for the encryption of attack reports and to sign said attack reports are stored directly in a system contract. Since only the creator of the system contract, *i.e.* the BloSS instance responsible for all the subnetworks which the system contract was registered for, can change fields of the smart contract, only they are able to change the public key. This means that building upon the secure ledger inherent in the Ethereum blockchain allows us to build a highly decentralized but still very secure signaling system.

5.3 Configuration

Configuration for all modules of the BloSS is supported through a simple INI file containing multiple categories with settings ranging from logging levels over intervals and thresholds for the Stalk and Pollen modules to network configuration data for Stalk.

Listing 5.1: Excerpt of the example INI configuration

```
[DEFAULT]
TIMESTAMP_FORMAT = %Y-%m-%d-%H:%M:%S

[BLOCKCHAIN]
HOST_ADDRESS = 127.0.0.1
PORT = 8545
RELAY_CONTRACT_ADDRESS = 0xDEADBEEFFEED
RELAY_SOURCE_FILENAME = relay.sol
SYSTEM_SOURCE_FILENAME = autonomous_system.sol
ACCOUNT_PASSPHRASE = password
ACCOUNT_UNLOCK_DURATION = 9999999
SYSTEM_CONTRACT_ADDRESS = 0xDEADBEEFFEED

[NETWORK]
SUBNETWORKS = ["192.168.1.0/24"]
ROUTER_IP = 192.168.1.1
```

```
ROUTER_MAC = 6C:3B:6B:51:1D:2D
OUT_PORTS = {"192.168.1.0/24":2,"192.168.2.0/24":3,"192.168.3.0/24":1}
ADDRESSES = {"123917682137029": {"192.168.1.2":"2C:4D:54:42:C5:E2",
                                 "192.168.1.3":"2C:4D:54:42:C3:E9",
                                 "192.168.1.4":"2C:4D:54:42:C6:52",
                                 "192.168.1.5":"2C:4D:54:42:C8:4F"}}
```

Listing 5.1 shows an excerpt of the example configuration bundled with the BloSS for new users. It is important to note that all configuration in the BloSS is entirely handled through the *config.ini* file which enables quick deployment of code changes to multiple ASes since the Python code of the BloSS is identical for each installation.

The *config.ini* file is read by a purpose-built configuration class implementing the standard Python ConfigParser but extending it to use an access scheme similar to 2D arrays, *e.g.* `config['BLOCKCHAIN']['RELAY_CONTRACT_ADDRESS']` to get the relay contract address.

5.4 Stalk

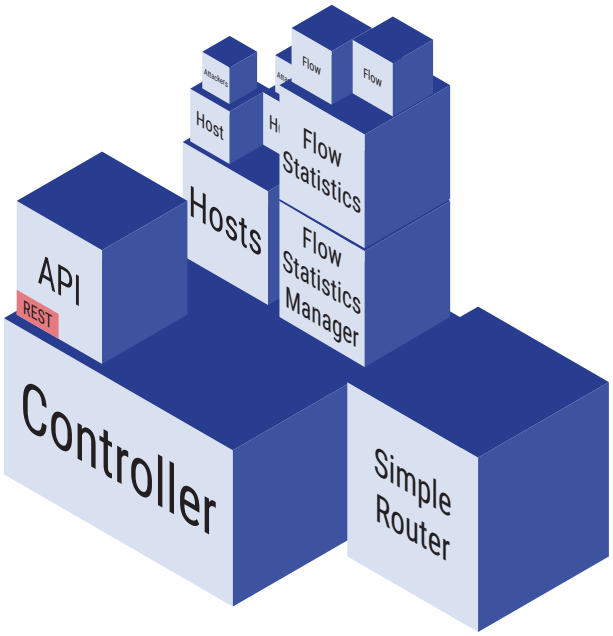

Figure 5.4: Classes of the Stalk module

The stalk module consist of two main classes directly communicating with the underlying networking infrastructure: The controller as well as the simple router. While the simple router just enables the correct forwarding of packets between ASes through the SDN switches, the controller analyzes flows from the same SDN switch and detects ongoing attacks.

To communicate with the SDN networking infrastructure, the Ryu library [54] is used which allows the development of SDN controllers directly in Python. Each SDN controller is realized as a Ryu App that can later be instantiated through the Ryu Manager.

5.4.1 Simple Router

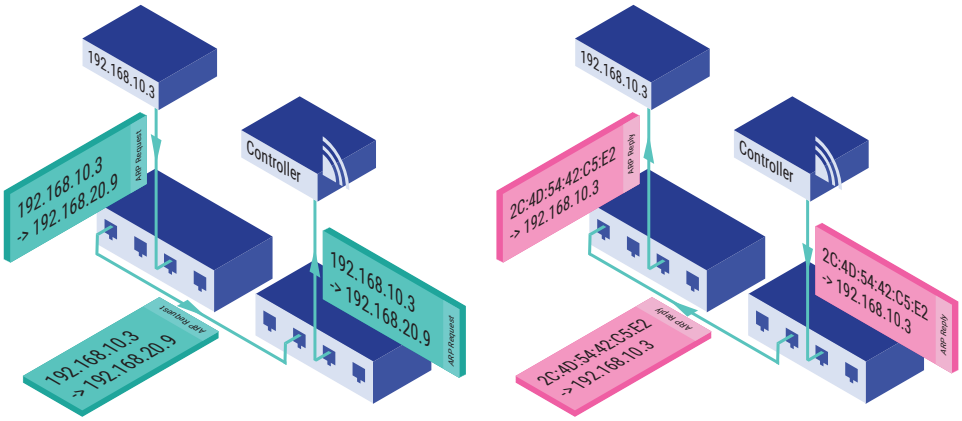

Figure 5.5: Sending and receiving ARP requests and replies is handled by the SDN switches

The simple router is necessary since the SDN infrastructure used in the demonstration system does not contain any logic for simple packet forwarding. The default rule in the Zodiac FX switches is to forward every packet to the SDN controller. In order to allow efficient routing of packets between the ASes, the simple router part of Stalk forwards Address Resolution Protocol (ARP) packets to the SDN controller managing a specific AS and receives in turn ARP replies containing the requested Media Access Control (MAC) address as illustrated in Figure 5.5. In order to be able to reply to these ARP requests, the simple router running on the controller maintains a mapping in *config.ini* of IP and corresponding MAC addresses for each host in the AS they are managing.

To generally allow forwarding of packets, the simple router sets up flows which specify how to handle packets coming and going to a specific IP address. Each flow clearly specifies an out-port to instruct the SDN switch on which port incoming packets should be sent out based on their origin and destination address. The out-port mapping is dependent on the physical cabling between the SDN switches as illustrated in Figure 5.1 and therefore needs to be configured by hand in the *config.ini* (*cf.* the "OUT_PORTS" entry in Listing 5.1) for the simple router to rely on.

The basic structure of the simple router is based on sample code by Toshiki [55] and extended with the ARP request handling and out-port assignment backed up by the pre-defined networking information in the *config.ini* configuration file.

5.4.2 Controller

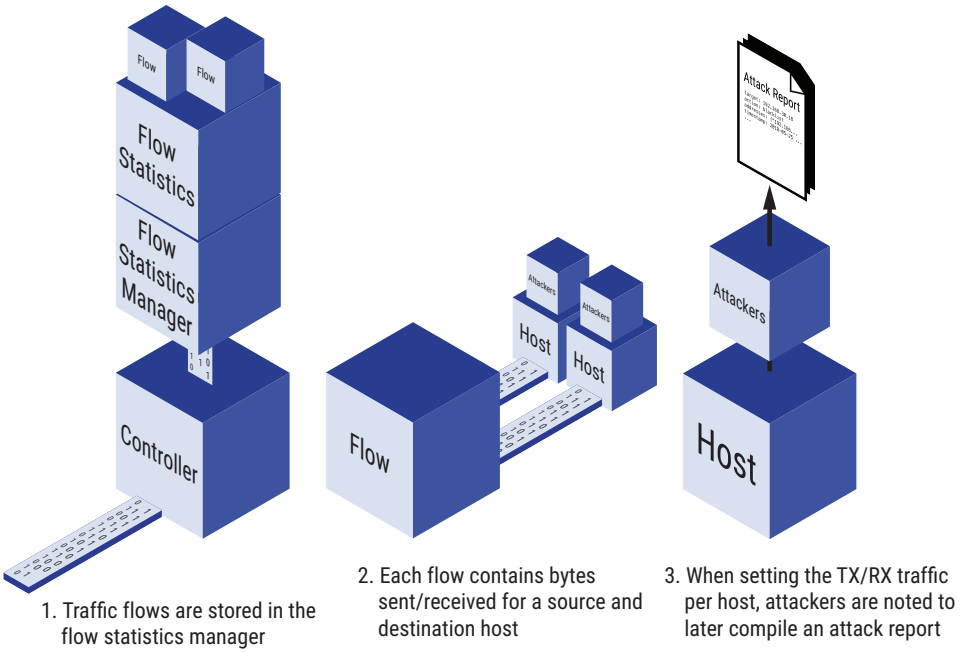

Figure 5.6: The Stalk controller analyzes the traffic and finds attackers

Analyzing traffic to find potential attackers is the main task of the Stalk controller. As illustrated in Figure 5.6, traffic flows received from the SDN switch are stored in the flow statistics manager. From there, bytes transferred and received for each flow are written to the corresponding source or destination host. The host objects are generated while initializing the Stalk controller and are comprised of all hosts managed by the controller as specified in *config.ini*.

Listing 5.2: Requesting flow statistics in Stalk controller

```
    def _request_flow_statistics(self):
        ...
        parser = datapath.ofproto_parser
        request = parser.OFPFlowStatsRequest(datapath)
        datapath.send_msg(request)
```

To receive detailed traffic flow information in the first place, the Stalk controller has to send flow statistics requests continuously as shown in Listing 5.2, which will cause the SDN switch to answer with the needed data. Access to SDN-specific functions works through decorators in Ryu, which are bound to OpenFlow events such as `EventOFPFlowStatsReply` to receive flow statistics replies from the SDN switch.

Whenever traffic volumes transferred or received are written to the corresponding source or destination host, the volume of traffic is checked against a threshold. If the traffic exceeds the threshold, the remote source of the traffic is noted as being a potential attacker.

As soon as the average traffic throughput volume for a host during a specified time window exceeds a threshold, the attackers are once again updated with the current traffic volumes and whether their throughput still exceeds the threshold and the resulting list of attackers is then used to compile an attack report. This attack report is then sent through the Python requests library directly to the BloSS REST API endpoint on the same machine.

The Stalk REST API serves as the connecting link between the BloSS module and Stalk and only consists of a simple Python Flask app maintaining the */api/v1.0/mitigate"* mapping to receive requests to block attackers from the BloSS module.

Listing 5.3: Blocking attackers based on an attack report

```
def block_attackers(self, attack_report):
    ...
    ofproto = datapath.ofproto
    if attack_report.action == "blackhole":
        instructions = [
            parser.OFPInstructionActions(
                ofproto.OFPIT_CLEAR_ACTIONS,
                []
            )
        ]
    else:
        instructions = []
    blocking_duration = (
        self._config['INTERVAL']
                    ['MAX_BLOCKING_DURATION_SECONDS']
    )
    mod = parser.OFPFlowMod(datapath=datapath,
                            command=ofproto.OFPFC_ADD,
                            priority=999,
                            idle_timeout=blocking_duration,
                            hard_timeout=blocking_duration,
                            match=match,
                            instructions=instructions)
    datapath.send_msg(mod)
```

This mitigation service of receiving attack reports and blocking the contained attacker addresses is the second task of the Stalk controller. Blocking works on a per-flow basis including a source address (the attacker) and destination address (the attack target). Listing 5.3 shows the relevant Python code to block a single flow. This is accomplished by creating a new flow with a very high priority of 999 which specifies to clear all other actions specified for this flow. The old flow specifying the packet forwarding with the corresponding out-port will therefore be overruled with this new flow definition. The blocking flow is however not permanent and will only exist for a pre-defined time window in order to allow the flow to normalize and avoid blocking benign traffic.

5.5 BloSS

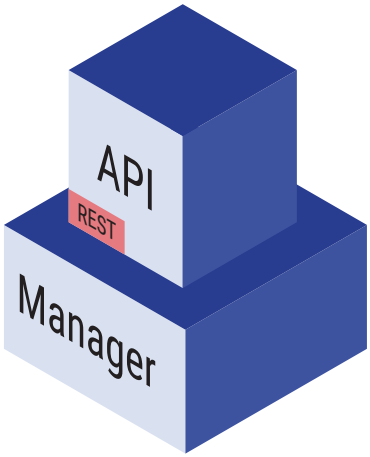

Figure 5.7: Classes of the BloSS module

The smallest module of the entire BloSS is the namesake module of the system: The BloSS module. As described in Section 4.1, the BloSS module has been decoupled from Stalk and Pollen to allow for a more network-agnostic system that would also facilitate a VNF-based encapsulation of the core parts of the BloSS.

The BloSS module consists of two classes as illustrated in Figure 5.7. The BloSS REST API receives attack reports from the Stalk module running on the same machine and posts these attack reports to the blockchain with the help of the Pollen module. It also maintains an API mapping to receive requests from the Stalk module to set a specific attack report to ”blocked” on the blockchain to signal that the report has been addressed.

The manager on the other hand periodically retrieves attack reports from the blockchain and sends the decrypted reports directly to the Stalk REST API to be blocked as discussed in Section 5.4.2.

5.6 Pollen

All data-exchange apart from basic networking handled by Stalk is taken care of by the Pollen module. Pollen therefore consists of a multitude of specialized classes as illustrated in Figure 5.8. These classes handle Ethereum access (PollenBlockchain), IPFS storage (PollenDatastore and PollenEncryption) and InfluxDB management (PollenDatabase). An additional helper class to handle attack reports (AttackReporting) provides simple parsing and processing capabilities to make sure attack reports are always formatted correctly and did not yet expire.

The PollenDatabase class is only used in order to enable the centralized storage of traffic information for visualization in Grafana [20]. It is important to note that PollenDatabase is not an integral part of the Pollen module or the BloSS as a whole and only serves as a

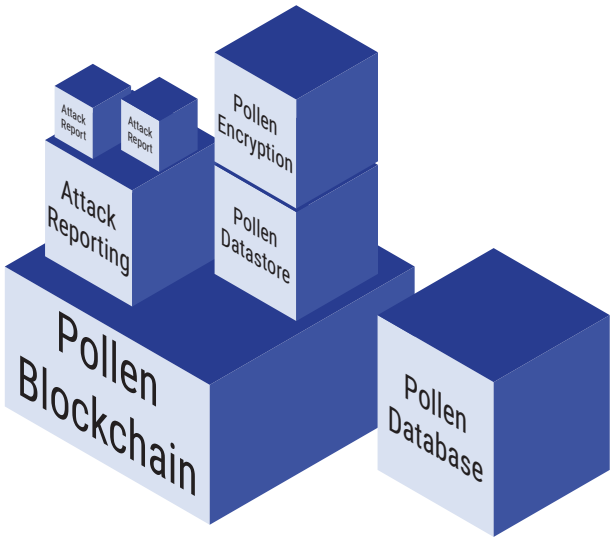

Figure 5.8: Classes of the Pollen module

debugging and demonstration tool since it would otherwise be complicated to gather the traffic information of multiple BloSS instances to figure out whether and were a problem exists. The centralized aspect of the InfluxDB where traffic information is stored would clearly defeat the goal of building a decentralized and scalable DDoS defense system which is why it is important to clearly state the demonstration-focused nature of PollenDatabase.

5.6.1 PollenBlockchain

PollenBlockchain is capable of automatically creating the system contracts upon initialization as illustrated in Figure 5.9. It first checks, whether a system contract address is already defined in *config.ini* and if this is not the case, it uses the solidity compiler [22] to create the solidity bytecode which is then deployed through PollenBlockchain. After deploying a system contract, it needs to be registered with the relay contract to specify which subnets the system contract is responsible for. This is accomplished with a direct transaction from PollenBlockchain to the relay contract.

All Ethereum-related transactions go through the Web3 Python library [25] which is connected to the RPC API of the geth Ethereum node running on the controller. Access to the RPC API is limited to localhost connections by default, which is why the geth Node needs to run on the same controller as the BloSS instance.

Apart from deploying the system contract, the PollenBlockchain class posts the public key used to encrypt and sign attack reports to Ethereum. This is done as the last step in the initialization and ensure, that the public key available on the blockchain always represents the correct key set up in the BloSS instance.

After all initialization work is done, the PollenBlockchain class is a passive entity that provides three main functions to other classes: Reporting and retrieving attack reports as well as marking attack reports as blocked in the system contract. Only the BloSS module uses the PollenBlockchain class, in order to maintain loose coupling between the modules.

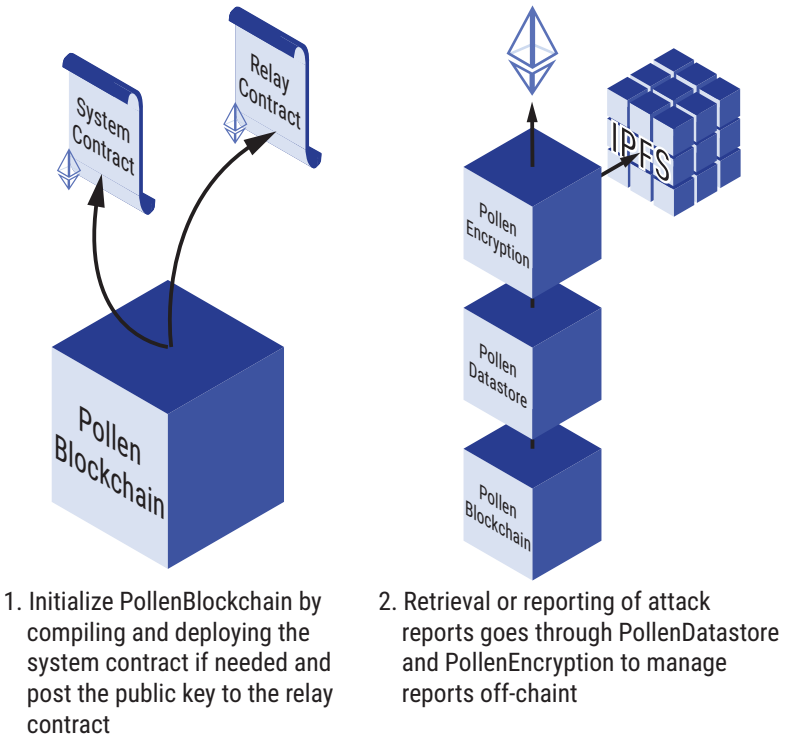

Figure 5.9: Duties of the PollenBlockchain class

5.6.2 PollenDatastore and PollenEncryption

To facilitate off-chain storage of attack reports, the PollenDatastore class provides the necessary ”store” and ”retrieve” methods to interface with IPFS. Since attack reports often contain sensitive information that could incriminate the attackers but more importantly the victims of large-scale DDoS attacks, the PollenEncryption class can be used.

The user can decide whether they want to use encryption or not by specifying it in the *config.ini* configuration file. However, if they opt out of using encryption, they won't be able to participate in a cooperative defense alliance with other BloSS instances encrypting their attack reports.

Encryption and decryption is built with the Python Cryptography library [8] and the choices for cryptographic algorithms as well as key lengths and other cryptographic details are based on an article by Colin Percival [40]. PollenEncryption uses both asymmetric cryptography through RSA with 2048 bit keys and symmetric cryptography through Fernet, which is essentially the Advanced Encryption Standard (AES) block cipher in Cipher Block Chaining (CBC) mode using a 128 bit long key [8].

Asymmetric encryption is used for two tasks in PollenEncryption: To encrypt the symmetric key as well as to cryptographically sign the unencrypted attack report with the private key of the sender. Signing the attack report with the private key allows the receiver to verify the authenticity of the attack report by using the public key available on the blockchain for cryptographic verification.

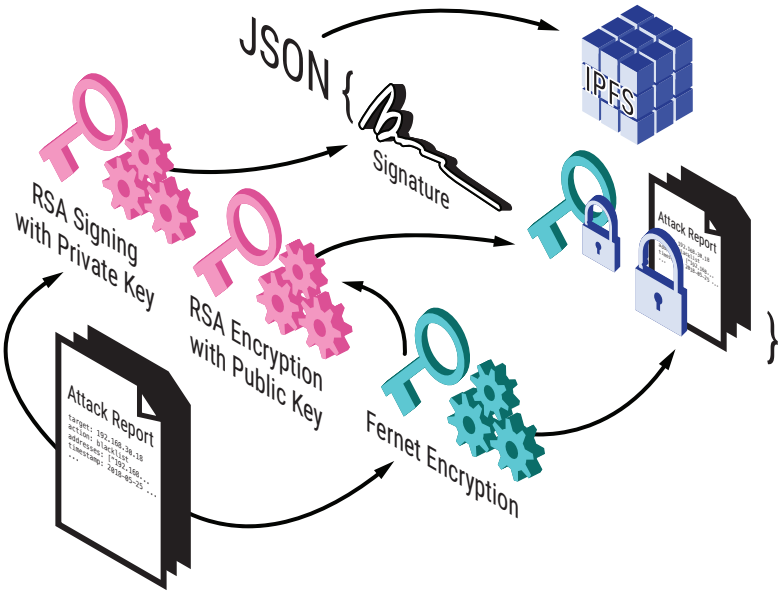

Figure 5.10: Encryption procedure for an attack report

Instead of directly encrypting the attack report through asymmetric encryption, the symmetric Fernet scheme is used. Asymmetric encryption is very useful since no secret key exchange has to occur, however it is not well suited to encrypt large amounts of data since the size of data to be encrypted cannot exceed the key size of 2048 bits[40].

The entire encryption procedure is depicted in Figure 5.10. After encrypting the attack report and symmetric key as well as signing the attack report, all three components, signature, encrypted symmetric key and encrypted attack report are stored in IPFS as a JavaScript Object Notation (JSON) object for easier handling through the Stalk and BloSS REST APIs.

Chapter 6

Evaluation

The evaluation of the re-engineered BloSS should reflect how well the system is suited to defend against high-volume DDoS attacks while keeping a small footprint per BloSS instance in order to minimize the resource requirements for the AS operators. In order to evaluate the system in these regards, two main aspects have been considered: The delay from the beginning of an attack until the attack has been completely blocked as well as the CPU usage of each BloSS instance throughout the entire mitigation.

6.1 Evaluation Setup

The BloSS has been evaluated on the demonstration system as described in Section 5.1 by utilizing the iperf network bandwidth measurement tool [53]. An instance of iperf is installed on the last compute node of AS 300 with IP address 192.168.30.18 and is listening for incoming iperf connections on UDP port 5000.

To start an attack, the following command is issued to all hosts from AS 100 and AS 200:

```
# iperf -c 192.168.30.18 -u -b 20m -t 10 -i 1 -p 5000
```

The remaining three hosts from AS 300 (192.168.30.15, 192.168.30.16 and 192.168.30.17) are idle throughout the attack since it is impossible to block traffic flowing from them to the target host 192.168.30.18. This is due to how the demonstration system is set up as described in Section 5.1.1. Only traffic between ASes goes through the SDN switches and can therefore be affected by the SDN controller. Intra-AS traffic does however go through the switch and router to which all compute nodes of an AS are connected. This is based on the limited number of ethernet ports on the Zodiac FX switches as outlined in Section 5.1.1

6.2 Blocking Delay

To evaluate the delay from the start of an attack to the attack being completely blocked by the cooperative defense, traffic statistics from InfluxDB were used. PollenDatabase, the database component of the Pollen module posts traffic volume information every second to the InfluxDB. To get the delay, only database entries with a bandwidth value of over 1 Mbit/s were considered since these correspond to the entries written during the attack. Since traffic information is written every second, the select statement in Listing 6.1 just counts the number of returned rows which correspond to the amount of time passed from beginning to end of the attack.

Listing 6.1: Select statement to determine the delay in seconds until an attack was blocked

```
SELECT count(mbps)
FROM "inboundtraffic"
WHERE "datapath_id" = '123917682137033' AND mbps > 1
```

This does of course only work if only a single attack in terms of volume is present in the database. To make sure this was the case, the database was routinely cleared before each attack to eliminate old attack volumes.

Table 6.1 shows delays recorded for 4 different bandwidths and over 10 attacks for each bandwidth. The bandwidth is set per compute node, which means at a bandwidth of 10 Mbit/s, a total attack volume of 80 Mbit/s is created and routed toward the target compute node.

Table 6.1: Delay until attacks with different bandwidths are blocked

Bandwidth/Delay (s)											**Average**
10 Mbit/s	34	34	28	20	28	27	26	37	23	19	27.6
20 Mbit/s	32	18	34	18	28	31	32	31	16	32	27.2
40 Mbit/s	34	25	39	24	31	35	25	29	31	24	29.7
100 Mbit/s	34	26	40	29	29	24	35	28	36	35	31.6

Complete evaluation data for each attack is available as part of the CD bundled with this thesis. It shows detailed information about each attack including timestamps and bandwidth per time step throughout the entire attack.

It is important to note that one of the biggest contributing factors to the delay is the block period of 5 seconds as described in Section 5.2. After sending an attack report, 5 seconds pass until the attack report becomes available to all BloSS instances. Since attack reports are based on subnetworks, a minimum of two reports need to be sent out in order to cover the two attacking subnetworks 192.168.10.0/24 and 192.168.20.0/24. If one of the attacking hosts is detected with a delay, another attack reports needs to be filed which consumes another 5 seconds.

6.3 CPU Load

The additional processing resources required to run the BloSS are an important metric to decide whether it is worth to tolerate the added strain on the CPU in the light of being able to mitigate DDoS attacks that could otherwise not be handled due to their distributed nature.

6.3.1 Collection

The evaluation in regards to CPU load consists of a full mitigation including all 8 attack hosts and 1 target host with a total attack volume of 160 MBit/s. The experiments are split into mitigations with encrypted attack reports and mitigations with encryption turned off entirely. With this differentiation, the added confidentiality provided through the encrypted attack reports can be contrasted with possibly increased CPU usages. To obtain the CPU usages of individual BloSS instances, the Python code in Listing 6.2 was used, which reads the CPU time of a process directly from */proc*. CPU time consumed is increasing throughout the lifetime of a process, so in order to determine the CPU usage throughout a specific time slot, the delta of CPU time consumed at the beginning and end of the time slot needs to be considered.

Listing 6.2: Python code to determine CPU time consumed by a process

```
def get_cputime(hz, pid):
    stats = check_output(['cat', '/proc/' + pid + '/stat'])
    stats_array = stats.split(' ')
    return (float(stats_array[13])
            + float(stats_array[14]))/hz
```

After determining the CPU usage of the BloSS instance, the value is written to a CSV file together with the timestamp. This is executed every second in order to be able to later correlate the CPU usage with the amount of inbound traffic on the target host stored on InfluxDB as outlined in Section 6.2.

6.3.2 CPU Usage Graphs

In order to account for variations such as delays in sending the attack traffic or other processes on the system using up available processing resources, 5 experiments with encryption turned on and 5 experiments without encryption have been carried out.

Figure 6.1 shows a graph detailing CPU usages for the controllers of all three ASes combined with the attack volume. Since the ASUS Tinkerboard single board computers are limited to 100 MBit/s throughput, the complete attack volume of 160 MBit/s is not visible on the graph. The choice for using a total volume of 160 MBit/s ensures that the inbound link of the target compute node is completely saturated to create a realistic, high-bandwidth DDoS attack scenario. A small peak in CPU usage for AS 300 right after

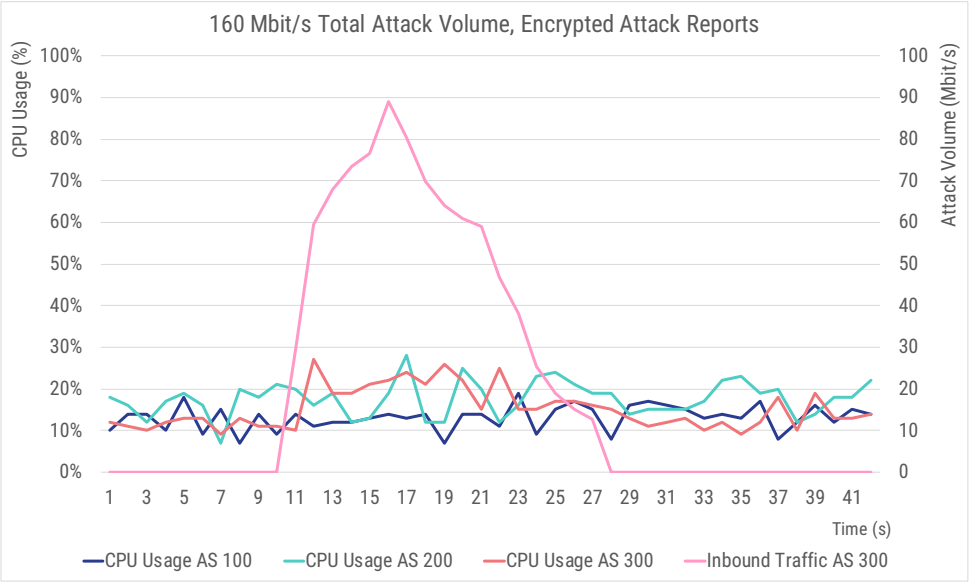

Figure 6.1: CPU usage evaluation with encryption turned on

the attack starts signifies the signaling of the attack to the blockchain. Shortly after, small peaks for the CPU usage of AS 100 and AS 200 can be seen, which indicate that the attack reports have been fetched from the blockchain, decrypted and acted upon.

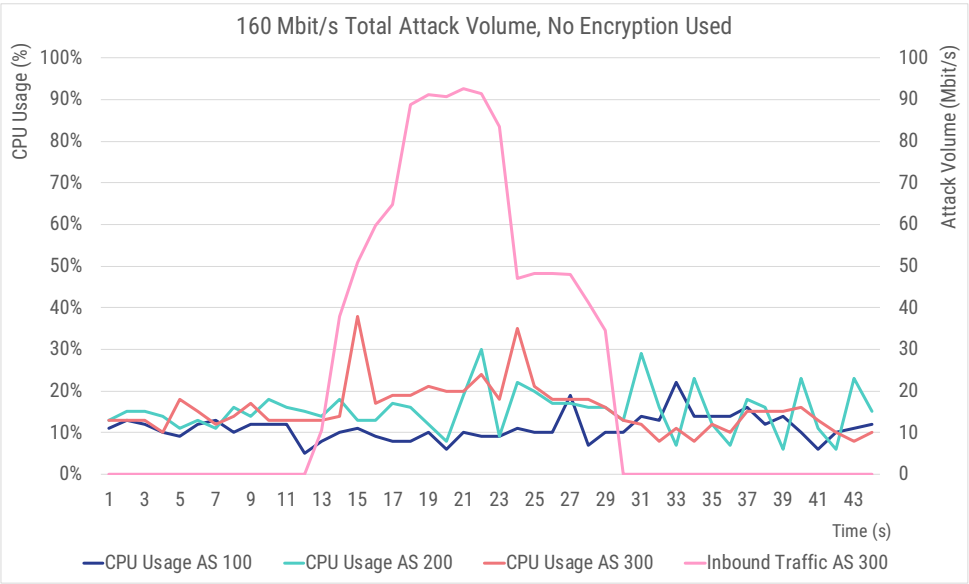

Figure 6.2: CPU usage evaluation without encryption

Figure 6.2 shows a run of the experiment with encryption turned off. This run is particularly interesting due to the two-stage signaling that was required to fully mitigate the attack. After the first signaling, only a reduction to around half the initial attack volume can be seen, however after a second round of signaling, the entire attack has been mitigated. It is also interesting to note that the CPU usage peaks indicating signaling for AS 300 are higher compared to Figure 6.1. This showcases the importance of multiple runs of the same experiment, to make sure that these differences do not directly lead to

conclusions that might be based on wrong assumptions.

6.3.3 Statistical Metrics

In order to better understand the variations in CPU usages between the two scenarios with and without encryption, minimum, maximum, average and median CPU usage across all ASes and all runs per experiment have been compiled into two tables. Each entry represents the average over 5 runs for the specific statistical indicator and AS. Table 6.2 shows statistics for the 5 runs with encryption turned on and Table 6.3 shows the other 5 runs with encryption turned off.

Table 6.2: CPU usage statistics across all 5 runs with encryption turned on

CPU Usage (%)/AS	**AS 100**	**AS 200**	**AS 300**
Minimum CPU Usage	5.8%	8.2%	8.8%
Maximum CPU Usage	21.2%	27.8%	34.4%
Average CPU Usage	13.2%	17.2%	16.3%
Median CPU Usage	13.4%	16.9%	15%

Table 6.3: CPU usage statistics across all 5 runs with encryption turned off

CPU Usage (%)/AS	**AS 100**	**AS 200**	**AS 300**
Minimum CPU Usage	4.8%	6.6%	8.4%
Maximum CPU Usage	18.2%	24.6%	34.6%
Average CPU Usage	10.5%	14.8%	15.8%
Median CPU Usage	10.3%	14.8%	14%

Detailed data collected throughout the experiments is included in the CD bundled with this report. There, graphs for each experiment together with the raw data in spreadsheet format consisting of CPU usage per AS and inbound attack volume for AS 300 are available. Furthermore, raw data for the statistics presented in Tables 6.2 and 6.3 are available as spreadsheets listing all values on which the statistics are based.

Chapter 7

Discussion

This chapter starts with the study of the evaluation results gathered in Chapter 6 to allow a differentiated view of the practicability and whether the re-engineered version of BloSS provides added values beyond the existing proof of concept implementation developed by Rodrigues *et al.* [46].

In a second part, the BloSS is contrasted with existing decentralized defense systems as presented in Chapter 2. This should provide the reader with a concluding idea about the unique characteristics that differentiate the BloSS from all other decentralized DDoS mitigation systems and makes a case for using blockchains as highly reliable signaling instruments.

7.1 Evaluation Results

The focus of the evaluation clearly lies on determining, whether the re-engineered BloSS excels as a practicably usable multi-domain DDoS mitigation system. This is important, since the core idea of the re-engineered BloSS is to provide a research platform on top of which a full-fledged Mitigation-as-a-Service (MaaS) offering could be realized. To enable this, the BloSS as the core framework needs to be sound in regards to providing the basic mitigation capabilities while maintaining a small enough footprint to allow for additional components and system to be built on top.

7.1.1 Blocking Delay Results

The most important observation from the evaluation data presented in Section 6.2 is the small amount of variation in regards to different bandwidths.

This is an important aspect of the system and goes to show that the BloSS is capable of mitigating attacks regardless of whether the attack volume accounts to more than 8 times the available throughput of the target system (in the case of the experiment with 100 MBit/s bandwidth per attacker) or only around 80% of the available throughput.

In addition to that, the average mitigation time of around 29 seconds over all experiments shows that the BloSS is a fast-acting mitigation solution capable of quickly diminishing even high-bandwidth threats. This speed is mostly courtesy of the 5 second block period as discussed in Section 5.2 as well as the fully automated nature of the demonstration system which does not include negotiation between target and mitigator. However, this is precisely why this short delay to block an attack is so important: As a research platform, the BloSS will eventually be extended with a payment based incentive as well as a reputation scheme to build a MaaS offering. If the underlying, barebones mitigation system would however already be slow to mitigate simple attacks, this would not be a good base to build a more complex system on top of.

7.1.2 CPU Usage Results

Apart from the ability of the BloSS to quickly mitigate attacks, it should not consume too many resources while doing so. The 10 experiments in Section 6.3 suggest that the highest degree of CPU usage occurs at the point in time when attacks are reported to the blockchain or retrieved by the mitigators. This can be observed from the small spikes in CPU usage in the two example graphs shown in Section 6.3.2.

This behavior is to be expected but the interesting aspect of the evaluation in Section 6.3 lies in the statistical metrics that show over 5 experiment runs each, that the difference in CPU usage between the BloSS instances fully relying on encryption is only around 2% higher than the other half of the experiments with encryption turned off. This is a clear indicator that encryption does not add considerable overhead to the BloSS and can therefore safely be left turned on, especially considering that encryption provides a high degree of confidentiality and allows to ensure the integrity of the attack reports through the verification of cryptographic signatures.

By collecting CPU usage information for a short period before and after the attack occurs, it can also be shown that the attack itself does not contribute to a considerable increase in CPU usage, with the exceptions of the short bursts to post attack reports and retrieve them from the blockchain. The baseline CPU usage of around 15% has to be attributed toward the Python programming language, which is an interpreted language, therefore creating a small overhead when running programs written in that language. Apart from this, the BloSS requires to periodically analyze traffic information as well as request attack reports from the blockchain, which both contribute toward the baseline CPU usage.

Since 15% of average CPU usage is not a negligible amount of processing resources consumed, this value could be improved by reducing the frequency of the aforementioned periodic tasks. This would however result in increased delays to mitigate ongoing attacks. Since the delay is an important factor in enabling an efficient incentive scheme later on, this trade-off does not seem reasonable and keeping the frequencies at the current high rate is therefore advisable.

7.2 Competition

The field of DDoS defense system contains an increasing number of competing approaches to solve the multi-domain DDoS mitigation problem. The BloSS is best comparable with the decentralized systems as presented in Section 2.2. These approaches mainly differ in the communication mechanism they employ.

While the DOTS architecture pioneered by the IETF is built on top of a purpose-built communication protocol specifically designed for the use in DDoS defense signaling [28], other approaches such as DefCOM base their communication mechanisms on existing overlay network technology to enable signaling in a peer to peer manner [37].

Compared to these two approaches, the BloSS is more akin to DefCOM than DOTS since it also builds on an existing system in the form of the Ethereum blockchain instead of developing a new communication approach from scratch. Leveraging the highly distributed nature and secure ledger aspects of blockchains allows the BloSS to securely and easily scale to the demand of a modern distributed DDoS defense system. DefCOM on the other hand relies on complex peer to peer message exchanges that are more prone to failure than the consensus-based blockchain system utilized by the BloSS.

By building the demonstration implementation of the BloSS with SDN and NFV-capability in mind, advantages of quick deployment inherent to competing systems like CoFence [44] or Bohatei [9] are already part of the re-engineered BloSS. With the modular aspect of the BloSS in mind, we are however not limited to SDN based networking infrastructures or only being able to provide the BloSS as a NFV-based solution but can adapt Stalk, the networking module of the BloSS to various infrastructures while still maintaining the simple, yet efficient RESTful interface between the BloSS module and Stalk.

Chapter 8

Final Considerations

In the course of this master thesis, the existing proof of concept implementation of a blockchain-based DDoS defense system was re-engineered to exhibit a modular and more secure structure. For this purpose, a demonstration implementation of the BloSS includes an SDN-based networking module called "Stalk", a data-exchange focused module called "Pollen" and a coordination module called "BloSS". In addition to the modularization and general re-structuring of the entire BloSS architecture, off-chain data storage powered by IPFS and secured through symmetric and asymmetric encryption schemes has been implemented.

The entire high-level design as well as the technical implementation of the re-engineered BloSS has been documented and all relevant aspects of the system have been discussed to allow the reader to understand the development process as well as the inner workings of the BloSS.

With the goal of building a research platform to enable the development of a true Mitigation-as-a-Service (MaaS) offering, the BloSS was evaluated in regards to practicability as well as performance aspects and discussed in the light of whether meets the requirements to serve as the basis for future research conducted in the field of blockchain-based DDoS mitigation systems.

As a research-focused contribution, four approaches to solve the key problem of providing an independently verifiable Proof-of-Mitigation are presented. These approaches serve as reference points for future research in enabling an efficient, payment-based incentives scheme as a core part of the aforementioned MaaS offering. Together with the re-engineered BloSS, these approaches aim to jump start a wave of innovative developments that could make the BloSS a leading strategy to efficiently counter high-bandwidth DDoS attacks.

8.1 Future Work

The demonstration implementation as discussed in Chapter 5 only provides an SDN-based implementation of Stalk. This is largely based on the SDN-focused demonstration system, however developing an alternative implementation of Stalk that would work with traditional networking infrastructures would make the BloSS truly network-agnostic instead of only providing the capability based on the modularization aspect of the networking component.

Due to hardware limitations of the single board computers as well as the Zodiac FX SDN switches used in the demonstration system (*cf.* Section 5.1), the maximum bandwidth of the system was limited to 100 MBit/s. Upgrading the demonstration system with better networking equipment could allow gigabit speeds and therefore allow attack simulations with even higher volumes to make sure the BloSS is capable of coping with these more realistic attacks.

Deployment and operation of the BloSS is still mostly a manual process, which limits adoption of the system as a research base since the steep learning curve of getting to know all subsystems involved in the BloSS and being able to run and maintaining them limits the attractiveness of using the BloSS. This could be addressed by setting up an automatic deployment system to allow one-click installation and instantiation of the BloSS.

Implementing a robust reputation and reward system as discussed by Andreas Gruhler [12] could enable the BloSS to only rely on a Proof-of-Mitigation for the incentives scheme until a base reputation has been established, for where on incentive payouts could directly be linked to the reputation of a mitigator.

Bibliography

[1] Mohammed Achemlal, Said Gharout, and Chrystel Gaber. Trusted platform module as an enabler for security in cloud computing. In *Network and Information Systems Security (SAR-SSI), 2011 Conference on*, pages 1–6. IEEE, 2011.

[2] Juan Benet. IPFS-Content Addressed, Versioned, P2P File System. *arXiv preprint arXiv:1407.3561*, 2014.

[3] Zdravko Bozakov and Panagiotis Papadimitriou. Autoslice: automated and scalable slicing for software-defined networks. In *Proceedings of the 2012 ACM conference on CoNEXT student workshop*, pages 3–4. ACM, 2012.

[4] Joao BD Cabrera, Lundy Lewis, Xinzhou Qin, Wenke Lee, Ravil K Prasanth, B Ravichandran, and Raman K Mehra. Proactive detection of distributed denial of service attacks using mib traffic variables-a feasibility study. In *Integrated network management proceedings, 2001 IEEE/IFIP international symposium on*, pages 609–622. IEEE, 2001.

[5] Brian Caswell, James C. Foster, Ryan Russell, Jay Beale, and Jeffrey Posluns. *Snort 2.0 Intrusion Detection*. Syngress Publishing, 2003.

[6] Victor Costan and Srinivas Devadas. Intel SGX Explained. *IACR Cryptology ePrint Archive*, 2016.

[7] PJ Criscuolo. Distributed denial of service, tribe flood network 2000, and stacheldraht ciac-2319, department of energy computer incident advisory capability (ciac). Technical report, UCRL-ID-136939, Rev. 1., Lawrence Livermore National Laboratory, 2000.

[8] Cryptography Developers. Python cryptography library. `https://cryptography.io/en/latest/`, 2017. [Online, accessed 2018-7-28].

[9] Seyed Kaveh Fayaz, Yoshiaki Tobioka, Vyas Sekar, and Michael Bailey. Bohatei: Flexible and elastic ddos defense. In *USENIX Security Symposium*, pages 817–832, 2015.

[10] Michal Feldman, Christos Papadimitriou, John Chuang, and Ion Stoica. Free-riding and whitewashing in peer-to-peer systems. *IEEE Journal on selected areas in communications*, 24(5):1010–1019, 2006.

[11] Thomer M Gil and Massimiliano Poletto. Multops: A data-structure for bandwidth attack detection. In *USENIX Security Symposium*, pages 23–38, 2001.

[12] Andreas Gruhler. A Reputation and Reward Scheme for a Cooperative, Multi-domain DDoS Defense. Master's thesis, University of Zurich, Department of Informatics, Communication Systems Group, Zurich, Switzerland, 2018.

[13] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. Peerreview: Practical accountability for distributed systems. *ACM SIGOPS operating systems review*, 41(6):175–188, 2007.

[14] John Hawkinson and Tony Bates. Guidelines for creation, selection, and registration of an autonomous system (as). Technical report, 1996.

[15] InfluxData. InfluxDB - Scalable datastore for metrics, events, and real-time analytics. `https://github.com/influxdata/influxdb`, July 2018. [Online, accessed 2018-7-27].

[16] John Ioannidis and Steven M Bellovin. Implementing pushback: Router-based defense against ddos attacks. In *NDSS*, volume 2, 2002.

[17] Michael Jakl. Representational state transfer, 2005.

[18] A John and T Sivakumar. Ddos: Survey of traceback methods. *International Journal of Recent Trends in Engineering*, 1(2):241, 2009.

[19] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[20] Grafana Labs. Grafana - The Open Platform for Analytics and Monitoring. `https://github.com/grafana/grafana`, July 2018. [Online, accessed 2018-7-27].

[21] Protocol Labs. Ipfs stream security transport (libp2p-secio). `https://github.com/libp2p/go-libp2p-secio`, 2018. [Online, accessed 2018-7-29].

[22] Stephan Mannhart. A custom debian stretch package for amd64 and armhf to install the solidity compiler solc v0.4.24. `https://github.com/savf/solc.deb`, June 2018. [Online, accessed 2018-7-28].

[23] Stephan Mannhart, Bruno Rodrigues, Eder Scheid, Salil S. Kanhere, and Burkhard Stiller. Toward Mitigation-as-a-Service in Cooperative Network Defenses. forthcoming.

[24] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[25] Piper Merriam and Jason Carver. Web3.py. `https://web3py.readthedocs.io/en/stable/`, 2018. [Online, accessed 2018-7-28].

[26] Jelena Mirkovic, Gregory Prier, and Peter Reiher. Attacking ddos at the source. In *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*, pages 312–321. IEEE, 2002.

[27] Jelena Mirkovic and Peter Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, April 2004.

[28] Andrew Mortensen, Flemming Andreasen, Tirumaleswar Reddy, Christopher Gray, Rich Compton, and Nik Teague. Distributed-Denial-of-Service Open Threat Signaling (DOTS) Architecture. Internet-Draft draft-ietf-dots-architecture-06, Internet Engineering Task Force, March 2018. Work in Progress.

[29] POA Network. Proof of authority: consensus model with identity at stake. `https://medium.com/poa-network/proof-of-authority-consensus-model-with-identity-at-stake-d5bd15463256`, Nov 2017. [Online, accessed 2018-7-27].

[30] A10 Networks. 2017 ddos of things survival guide. 2017. [Online, accessed 2018-7-24].

[31] Northbound Networks. Zodiac fx user guide. `https://zodiac-fx-germany.de/ZodiacFX_UserGuide_0216.pdf`, Feb 2016. [Online, accessed 2018-7-27].

[32] NFV White Paper. Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. Issue 1. October 2012.

[33] Hung Nguyen, Bipeen Acharya, Radoslav Ivanov, Andreas Haeberlen, Linh TX Phan, Oleg Sokolsky, Jesse Walker, James Weimer, William Hanson, and Insup Lee. Cloud-based secure logger for medical devices. In *Connected Health: Applications, Systems and Engineering Technologies (CHASE), 2016 IEEE First International Conference on*, pages 89–94. IEEE, 2016.

[34] Kaname Nishizuka, Liang Xia, Jinwei Xia, Dacheng Zhang, Luyuan Fang, Christopher Gray, and Rich Compton. Inter-organization cooperative DDoS protection mechanism. Internet-Draft draft-nishizuka-dots-inter-domain-mechanism-02, Internet Engineering Task Force, December 2016. Work in Progress.

[35] Nokia. Trusted nfv systems.

[36] Marian Oancea. Ethereum network stats official repository. `https://github.com/cubedro/eth-netstats`, December 2016. [Online, accessed 2018-7-29].

[37] George Oikonomou, Jelena Mirkovic, Peter Reiher, and Max Robinson. A framework for a collaborative ddos defense. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, pages 33–42. IEEE, 2006.

[38] OpenDaylight. OpenDaylight: A Linux Foundation Collaborative Project, 2013.

[39] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[40] Colin Percival. Cryptographic right answers. `http://www.daemonology.net/blog/2009-06-11-cryptographic-right-answers.html`, Jun 2009. [Online, accessed 2018-7-28].

[41] Ronald Perez, Reiner Sailer, Leendert van Doorn, et al. vtpm: virtualizing the trusted platform module. In *Proc. 15th Conf. on USENIX Security Symposium*, pages 305–320, 2006.

[42] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, Oakland, CA, 2015. USENIX Association.

[43] Gregor N. Purdy. *Linux iptables - kurz & gut.* O'Reilly, 08 2004.

[44] Bahman Rashidi and Carol Fung. CoFence: A Collaborative DDoS Defence Using Network Function Virtualization. In *12th International Conference on Network and Service Management (CNSM 16)*, October 2016.

[45] Sowmya Ravidas, Shankar Lal, Ian Oliver, and Leo Hippelainen. Incorporating trust in nfv: Addressing the challenges. In *Innovations in Clouds, Internet and Networks (ICIN), 2017 20th Conference on*, pages 87–91. IEEE, 2017.

[46] Bruno Rodrigues, Thomas Bocek, and Burkhard Stiller. Enabling a Cooperative, Multi-domain DDoS Defense by a Blockchain Signaling System (BloSS). In *Demonstration Track*, pages 1–3, Singapore, Singapore, Oct 2017. IEEE.

[47] Anbang Ruan and Andrew Martin. Repcloud: Attesting to cloud service dependency. *IEEE Transactions on Services Computing*, 10(5):675–688, 2017.

[48] Rishikesh Sahay, Gregory Blanc, Zonghua Zhang, and Hervé Debar. Towards autonomic ddos mitigation using software defined networking. In *SENT 2015: NDSS Workshop on Security of Emerging Networking Technologies*. Internet society, 2015.

[49] Salanfe. Setup your own private proof-of-authority ethereum network with geth. `https://hackernoon.com/9a0a3750cda8`, Feb 2018. [Online, accessed 2018-7-29].

[50] Bruce Schneier and John Kelsey. Cryptographic support for secure logs on untrusted machines. In *USENIX Security Symposium*, volume 98, pages 53–62, 1998.

[51] Ming-Wei Shih, Mohan Kumar, Taesoo Kim, and Ada Gavrilovska. S-nfv: Securing nfv states by using sgx. In *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 45–48. ACM, 2016.

[52] Mininet Team. Mininet: An instant virtual network on your laptop (or other pc). *Google Scholar*, 2012.

[53] Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, and Kevin Gibbs. iperf: Tcp/udp bandwidth measurement tool. 01 2005.

[54] FUJITA Tomonori. Introduction to ryu sdn framework. *Open Networking Summit*, 2013.

[55] Tsuboi Toshiki. Ryu Simple Forward. `https://github.com/ttsubo/simpleRouter/blob/master/ryu-app/blog/article_02/simpleForward.py`, January 2014. [Online, accessed 2018-7-28].

[56] Gavin Wood. Devp2p wire protocol. `https://github.com/ethereum/wiki/wiki/libp2p-Whitepaper`, 2014. [Online, accessed 2018-7-29].

[57] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. `https://goo.gl/LG7adX`, Jan 2016. [Online, accessed 2018-7-27].

[58] Saman T. Zargar, James Joshi, and David Tipper. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. *IEEE Communications Surveys Tutorials*, 15(4):pp. 2046–2069, Fourth 2013.

[59] Xuan Zhou, Rongpeng Li, Tao Chen, and Honggang Zhang. Network slicing as a service: enabling enterprises' own software-defined cellular networks. *IEEE Communications Magazine*, 54(7):146–153, 2016.

Abbreviations

DDoS	Distributed Denial of Service
MaaS	Mitigation-as-a-Service
CAPEX	CAPital EXpenditures
OPEX	OPErating EXpenditures
BloSS	Blockchain Signaling System
D-WARD	DDoS netWork Attack Recognition and Defense
MULTOPS	MUlti-Level Tree for Online Packet Statistics
MIB	Management Information Base
NMS	Network Management System
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
DOTS	Distributed-Denial-of-Service Open Threat Signaling
NFV	Network Function Virtualization
SDN	Software-Defined Networking
SLA	Service Level Agreement
VM	Virtual Machine
TPM	Trusted Platform Module
SGX	(Intel) Software Guard Extension
OS	Operating System
TSecO	Trusted Security Orchestrator
VIM	Virtual Infrastructure Manager
AMD	Advanced Micro Devices
NSaaS	Network-Slicing-as-a-Service
vSDN	virtualized Software-Defined Networking
SaaS	Software-as-a-Service
PaaS	Platform-as-a-Service
IaaS	Infrastructure-as-a-Service
IPFS	Inter Planetary File System
PoA	Proof-of-Authority
PoW	Proof-of-Work
ARP	Address Resolution Protocol
MAC	Media Access Control
AES	Advanced Encryption Standard
CBC	Cipher Block Chaining
JSON	JavaScript Object Notation

Glossary

BloSS The Blockchain Signaling System allows multi-domain DDoS defense by leveraging the Ethereum blockchain as a signaling medium.

DDoS According to Mirkovic *et al.* [26], a distributed denial-of-service attack "is comprised of packet streams from disparate sources. These streams converge on the victim, consuming some key resources and rendering it unavailable to legitimate clients" [26]

Mitigator The entity operating a mitigation system and therefore providing a mitigation service to a third party.

Autonomous System According to the RFC 1930 definition, an autonomous system "is a connected group of one or more IP prefixes run by one or more network operators which has a SINGLE and CLEARLY DEFINED routing policy."[14]

List of Figures

List of Tables

Appendix A

Toward Mitigation-as-a-Service in Cooperative Network Defenses

Toward Mitigation-as-a-Service in Cooperative Network Defenses

Stephan Mannhart, Bruno Rodrigues, Eder Scheid, Salil S. Kanhere*, Burkhard Stiller
Communication Systems Group CSG, Department of Informatics IfI, University of Zurich UZH, Switzerland
E-mail: stephan.mannhart@uzh.ch, [rodrigues,scheid,stiller]@ifi.uzh.ch
**Networked Systems and Security Group NetSyS, UNSW Sydney, NSW 2052 Australia*
E-mail: salil.kanhere@unsw.edu.au

***Abstract*—Distributed Denial-of-Service (DDoS) attacks are by design highly decentralized and therefore hard to defend against. By utilizing a decentralized, multi-domain, cooperative defense mechanism, it is possible to combine software and hardware capabilities to effortlessly mitigate large scale attacks. Cooperative defense systems face many challenges, such as deployment complexity due to high coordination overhead, reliance on trusted and stable channels for communication and the need for effective incentives to bolster cooperation among all involved parties. In particular, incentives are the key to ensure successful deployment of a "Mitigation-as-a-Service (MaaS)" for cooperative defense systems. This paper discusses the critical issue of providing a proof of the effectiveness of a cooperative defense mitigation, considering four state-of-the-art solutions toward an independently verifiable proof of mitigation. A qualitative analysis of these approaches across 9 dimensions shows that none satisfy all requirements due to the inherent trade-offs between practicability and security. As a result, it is identified that the issue of authenticating the underlying network flows remains unsolved.**

I. Introduction

The growing threat of Distributed Denial-of-Service (DDoS) attacks requires novel, fast-acting countermeasures. Mitigating large-scale DDoS attacks at the attack-traffic target is infeasible due to the overwhelming bandwidth of these attacks. A mechanism that allows multiple Autonomous Systems (AS) to mitigate the attack near its source within the managed address space of each AS can be an effective solution. For instance, it allows to combine the detection/mitigation capabilities of the cooperative entities, reduce the detection/mitigation overhead in a single entity, and block malicious traffic near its source. Enterprises targeted by DDoS attacks could subscribe to such a cooperative defense service by paying a fee to be protected from future threats. However, cooperative defense systems face three main challenges as outlined in [12], which include deployment complexity due to high coordination overhead, reliance on a trusted and stable channel for communication and the effective use of incentives to bolster cooperation among involved parties.

Incentives are necessary to cover CAPital EXpenditures (CAPEX) to set up communication infrastructures including additional hardware and software acquisition costs, and OPErating EXpenditures (OPEX) are incurred as soon as a mitigation service is in use. A possible solution to cover these expenditures is by passing them on to the service customer. Service fees paid by the customer can then be used as incentives among ASes involved in the cooperative defense to motivate the collaborative behavior.

However, the use of complex incentive schemes would lead to even higher complexity of operation, corresponding to an increase in subscription fees to cover OPEX and CAPEX. Incentive payout therefore needs to be based on an automated mitigation mechanism that can be verified independently to simplify and streamline the entire process. Additional costs related to delays and negotiations between ASes to verify whether the service was performed as agreed could then be avoided.

Avoidable costs include additional delays and negotiations between ASes after a mitigation has been conducted to verify whether the service was performed as agreed. However, since the effectiveness of a mitigation task needs to be manually verified by the target AS by examining logs and changes in attack traffic, an independently verifiable and automated mitigation proof is essential for the incentive scheme. Hence, automatic checks of the effectiveness of a mitigation, and subsequently, automatic payouts of incentives between ASes could be performed.

This paper discusses possible approaches to provide such an independently verifiable mitigation-proof and the challenges associated with each approach. We show that no satisfiable solution can be provided since a trade-off exists between practicability and security of each approach. In this regard, most of the discussed approaches are not clearly separable since they combine similar concepts and even a possible combination of these approaches would lead to an increase in complexity and lack of scalability. In addition to that, all of the approaches fail to include the integrity of the underlying network flows which leaves room for tampering and falsification of the mitigation proof. The literature research performed did not indicate other approaches which might address differently the core challenges as shown with the approaches presented in this paper.

This paper is organized as follows: Section II describes the relevance of a mitigation proof in the context of a cooperative defense. Section III presents approaches toward an automated solution, and in Section IV these approaches are compared. Section V concludes the work.

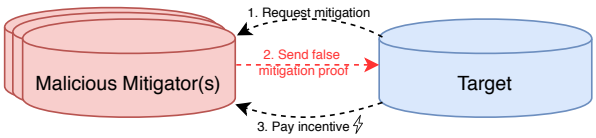

Figure 1. False-reporting of mitigation proof in a cooperative defense

II. Mitigation in a Cooperative Defense

The underlying process of a DDoS mitigation service involves multiple ASes collaboratively countering the DDoS attack. As soon as an attack is detected at an AS T (Target), it contacts ASes participating in the alliance to request mitigation. Then, an AS M (Mitigator), managing the address range responsible for the attack has the choice of accepting the mitigation request and providing the service by filtering the attack traffic.

A proof-of-mitigation (*cf.*, Figure 1) needs to exist to convey service completion to AS T in a way that will clearly confirm that a cooperative attack request has been successfully mitigated. Thus, upon acknowledgment of this proof, an incentive can directly be paid by AS T to AS M, which completes the process.

The crucial element of this exchange is the ability of an AS M to provide a proof-of-mitigation that satisfies the aspects of reproducibility, tamper-evidence and timeliness. If such a proof is not available right after completion of the mitigation, the other party will withhold the incentive payout and the overall service becomes unusable. Manual verification of a mitigation proof is also not feasible, due to the strict time constraints required to provide a fast-acting mitigation service able to counteract large-scale DDoS attacks. The timeliness aspect does therefore also include the aspect of being able to automatically verify the proof during the available time-window and excludes any user interaction in order to be efficient.

If the mitigation proof is falsified, the process fails, and the mitigation service cannot be provided, or the target AS erroneously pays the incentive since it failed to realize the falsification of the mitigation proof. Both scenarios would lead to a breakdown of the mitigation service offering since an inherent trust between both parties would be required instead of being able to rely on a verifiable proof.

For a qualitative discussion of the individual approaches presented in this paper, metrics are based on the scheme proposed by Zargar *et al.* [12] and focus on the deployment complexity as well as scalability while adding security related metrics not present in [12].

These additional metrics can provide an important overview of the presented approaches since providing a successful proof of mitigation is largely dependent on the security of the system generating the proof as well as the proof itself.

1) **Confidentiality**: Describes how effective rules and measures are to protect both the mitigation proof as well as the mitigation system from unauthorized access.
2) **Integrity**: Defines the level of trustworthiness of the proof generated by the given approach in regards to accuracy and tamper-resistance.
3) **Availability**: Relates to the availability of the mitigation proof to authorized parties.
4) **Reproducibility**: Describes the ability to reproduce the proof through replay by a third party.
5) **Tamper-Evidence**: Discloses the effort necessary for changing the proof of mitigation to reflect an alternate reality.
6) **Timeliness**: Defines the ability to provide an automatically verifiable proof within a pre-specified time-frame while adhering to all security requirements.
7) **Deployment Complexity**: Defines the additional resources required to deploy the approach.
8) **Scalability**: Relates to the adaptability of the approach in regards to a large-scale DDoS defense scenario with high-bandwidth attacks.
9) **Service Model**: Defines the cloud service model the approach most closely relates to.

III. Approaches Toward Mitigation Proofs

This Section presents approaches toward MaaS providing a short description overviewing their functioning and discussing their advantages and disadvantages.

A. Marketplace of Mitigation VNFs

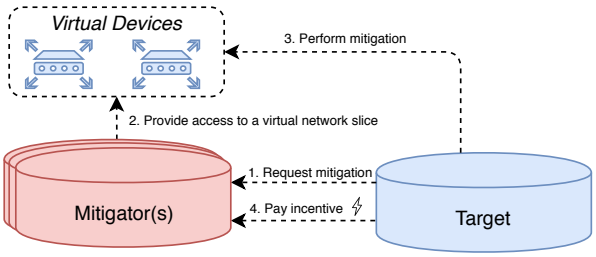

Figure 2. Mitigation device based on VNF available in a trusted marketplace

*1) **Description**:* Network Function Virtualization (NFV) [4] can provide an efficient solution to the deployment conundrum by allowing the mitigation system to be encapsulated as Virtualized Network Functions (VNF), which can directly be deployed on commodity hardware running on-site at the AS. Thus, to support a simple deployment of this approach, a VNF "marketplace" could be built as a platform of VNFs for all ASes involved in the cooperative defense. The mitigator AS would load the VNF directly from the marketplace to provide the mitigation service. This ensures that the VNF image used to conduct the mitigation is known

by all ASes and its integrity can be checked by comparing a hashed checksum of the image to a stored value on the marketplace. The marketplace also allows logging of access to the VNFs by the individual AS. Local caching of VNFs can be used to avoid increased load on the VNF marketplace.

*2) **Advantages**:* The high degree of isolation of this approach is a clear advantage. VNFs contain the minimal code necessary to conduct the mitigation and can therefore directly contribute to a certain degree of trust by implicitly ensuring that the correct code is executed. Running the mitigation inside VNFs also eases the deployment for ASes interested in offering such a service since they can run the VNF without any additional setup on supported hardware. Running VNFs only requires the necessary infrastructure for data plane control and can even be accomplished without directly using Software-Defined Networking (SDN) as outlined in [4].

*3) **Disadvantages**:* Encapsulating the mitigation service inside VNFs does not guarantee that the AS mitigator will not be able to tamper with the VNF itself. This means that the act of deploying the mitigation service through VNFs alone is not a reliable proof of mitigation and the AS requesting the service still needs to trust that the mitigator will only run untampered VNFs directly from the VNF marketplace. Even if the VNF has not been tampered with, network flows on which the mitigation will be conducted could be manipulated before they reach the VNF which could lead to a minimized or non-existent workload for the mitigator while still receiving the incentive payout from the AS requesting the service.

B. Trusted Computing

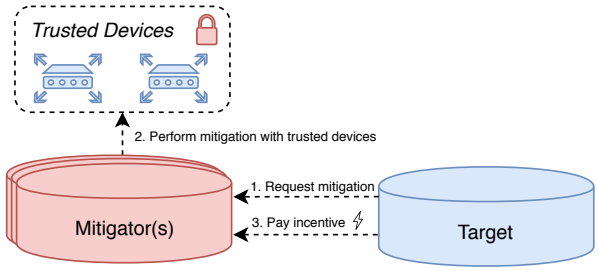

Figure 3. Mitigation devices are based on Trusted Platform Modules (TPM)

*1) **Description**:* A Trusted Platform Module (TPM) allows the secure storage of verification hashes in the on-chip Platform Configuration Registers (PCR) that can be used to attest a secure boot chain from BIOS over Bootloader, OS up to the Hypervisor [6]. Thus, code running inside the VM can be partly isolated by using technologies such as Intel Software Guard Extension (Intel SGX) which provides a secure enclave to store application states as proposed by Shih *et al.* [11].

To further extend the chain of trust up to the VNF itself, approaches like ”vTPM” [7] aim to fill this gap with a virtualized TPM instance. However, by virtualizing the hardware-based trust, a trade-off arises where the end-system gains a level of additional protection but is still not as secure as it could be with full hardware-based trusted computing. Another solution is remote attestation, where platform- as well as VNF-states are compared with known good values by a remote system. Traditionally, remote attestation was limited to non-virtualized systems and would therefore not directly apply to a VNF-based approach.

Ravidas *et al.* [9] propose a remote attestation server that goes beyond the platform-only integrity check and allows for full attestation of VNFs through image integrity checks. This is accomplished by introducing an external Trusted Security Orchestrator (TSecO) [9] which will receive VNF launch requests from a modified Virtual Infrastructure Manager (VIM). The decision to allow a launch request can then be based on whether the checksum for the VNF is featured in a whitelist published on the marketplace. These VNF image integrity checksums can later directly be used to check against known good values for remote attestation.

*2) **Advantages**:* The full chain of trust from hardware to the VNF guarantees that only the code that is approved by the cooperative defense can be run on the designated system. This guarantee combined with the marketplace that transparently provides the actual mitigation VNF creates a cooperative defense in which mitigation requests are always handled by known and trusted VNFs without the need of trusting the operator responsible for the AS providing the mitigation service.

*3) **Disadvantages**:* The biggest drawback of using a trusted computing approach to guarantee the integrity of the mitigation system are the strict hardware requirements. The TPM is a feature available only as a standalone chip or as a solution integrated into the motherboard but it does not come pre-installed on all systems, which greatly limits the potential of a DDoS mitigation system based on this technology. The same is true for Intel SGX, which is directly integrated into many Intel CPUs but not available for all Intel CPUs and is unavailable on competing products like AMDs CPUs.

Due to the nature of a trusted computing environment, the operator of the mitigating AS also gives up full control over their system. The policies enforced by the trusted computing environment will no longer allow the operator to run any VNF on the system equipped with a TPM since only whitelisted VNFs will be allowed in order to fully enforce the chain of trust. An additional problem similar to the NFV approach without TPM is the lack of control over the underlying networking infrastructure. The mitigator could once again change the network flow to the system running the mitigation VNF and lead it to believe that it is seeing all the traffic while in reality, parts or all of the

attack traffic have been rerouted and no mitigation seems to be required anymore. This would directly prompt AS T to initiate the incentive payout since the mitigation seems to have been concluded and all trusted computing related checks were passed.

C. Secure logging

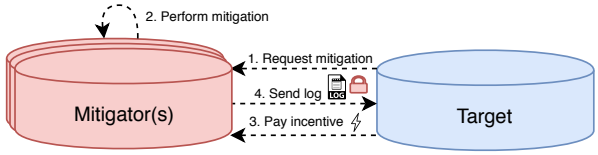

Figure 4. Secure Logging

*1) **Description**:* The main challenge in providing an independently verifiable mitigation proof is the reliance on the underlying networking infrastructure. Mitigating a DDoS attack by filtering traffic for example through blackholing requires the mitigator to have access to the networking infrastructure and to trust that the traffic represented by the infrastructure is the actual traffic passing through the mitigators AS. In the collaborative defense scenario, AS M in the role of the mitigator has full control over their underlying networking infrastructure which allows them to generate a proof of mitigation from the available traffic data.

This proof could consist of a detailed network log showing the effect of the mitigation as a reduction in the attack traffic from the attack source to the DDoS target. To secure this log, a scheme as presented in [10] could be used which utilizes authentication keys for the entire log as well as individual log entries together with a hash-chain to enable discovery of tampering attempts. This kind of secure logs have successfully been used to build various cloud-based systems with a high degree of accountability, such as [5] where secure logs from a medical device have been stored while maintaining tamper evidence. To further secure this log, the trusted computing approach discussed in Section III-B can be leveraged by utilizing both TPM as well as Intel SGX which has also been demonstrated by [5]. Intel SGX has specifically been proposed as a good extension for NFV integrity by [11] since it allows to protect specific application states inside the secure enclave provided by Intel SGX. In our case, storing the traffic logs inside that enclave would add a layer of security on top of the cryptographic methods proposed by Schneier *et al.* [10].

Since the log file has been created on an isolated system, checking the integrity of the log through a third party becomes an important aspect of its overall credibility. Haeberlen *et al.* [3] proposed a remote audit to ensure the correct operation of a remote system. This works similar to remote attestation as outlined in Section III-B1 but focuses on deterministic log files instead of binaries running on a system. By remotely replaying system executions and comparing the resulting log files to the logs obtained from the remote system, the correct behavior of the system under test can be determined.

*2) **Advantages**:* Reducing the mitigation proof to the log files minimizes the complexity of the overall proof. Checking the correctness of the log suffices to establish the success of the mitigation conducted by AS M. Together with the trusted computing schemes discussed in Section III-B1, secure logging requires no additional trust in the mitigation AS and the remote audit helps to cover any trust issues presented by the logging process itself.

*3) **Disadvantages**:* Protecting log files from tampering requires that they be stored securely, for example in tamper-proof hardware such as Intel SGX enclaves. This presents a similar disadvantage as with the trusted computing approach due to the requirement of specialized hardware. The secure enclave itself also needs to be protected by a chain of trust including all stages of the boot process right up to the operating system. To enable this, a TPM needs to be present in the system, which limits the hardware choice for a mitigation system even further.

Also, capturing the mitigation of a high-volume DDoS attack through network traffic results in large log files. Transferring these log files for remote auditing or to be used as the mitigation proof introduces additional delays in the mitigation process due to their large size. Storing logs to keep track of past mitigations also becomes a burden due to the immense disk space requirements.

D. Network Slicing

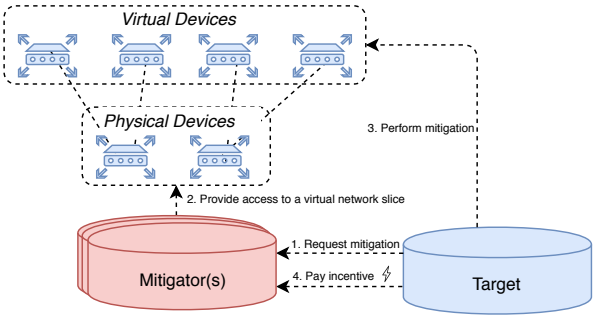

Figure 5. Virtual Slice

*1) **Description**:* Recent advances in network virtualization technologies and Software-Defined Networks (SDN) can be leveraged to realize network slicing as a service [13] for mitigation purposes. By requesting mitigation services from AS M, the target AS T gains access to a virtualized network slice. Based on the IP addresses provided with the mitigation request, the slice can be configured to only provide access to observe the flows of the attacking IP addresses.

Slice generation could be accomplished with "AutoSlice" proposed by Bozakov *et al.* [1] which creates on-demand virtualized SDN networks (vSDN)[1]. These vSDNs can be controlled by standard SDN controllers which would allow AS T to directly perform the mitigation on the target system.

*2) **Advantages**:* This approach shifts the requirements of the mitigation proof. There is no need for providing a direct proof that the mitigation has been carried out, since AS T has complete control over the whole process. The burden of providing the proof is lifted from AS M and the payout of the incentive from AS T can directly occur after the mitigation has been performed.

Creation of slices can be directly coupled to the mitigation request since the required information about relevant IP addresses to observe and control is directly provided together with the mitigation request.

*3) **Disadvantages**:* AS M has to give up full control of their networking infrastructure by providing the network slice as a service and AS T has to trust that the slice represents the portion of infrastructure of interest.

The implementation would also have a strong hardware as well as software requirement if realized with the AutoSlice[1] architecture: The networking infrastructure needs to be SDN-based and additional commodity servers with Open vSwitch installations are required to circumvent the constraint of limited flow-table sizes presented by most OpenFlow hardware switches[1].

IV. Discussion

Table I
Qualitative comparison of approaches toward MaaS

	NFV	Trusted Computing	Secure Logging	Network Slicing
Security				
1. Confidentiality	Low	Medium	Low	Low
2. Integrity	Low	High	High	Low
3. Availability	High	Medium	High	Medium
4. Reproducibility	High	High	Low	High
5. Tamper-Evidence	Low	Medium	Medium	Low
6. Timeliness	High	High	Low	Medium
Practicability				
7. Deployment Complexity	Low	High	High	High
8. Scalability	High	Low	Low	Low
Scope				
9. Service Model	SaaS	SaaS	PaaS	IaaS

Table I shows that no single approach satisfactorily addresses the trade-offs between security and practicability and could, therefore, be used by itself for an independent trustless mitigation service. NFV and Network Slicing have similar characteristics with respect to security due to their virtualized nature. While NFV virtualizes a single function in the network, Network Slicing aims to deliver a portion of the network infrastructure as a service. To accomplish this, approaches like AutoSlice [1] help to automate the creation of on-demand slices per mitigation request. However, the infrastructure requirements such as the need for SDN based networking for the automatic creation of vSDNs, increases the associated deployment complexity, thus limiting the applicability of this approach. An NFV-based approach on the other hand, presents a lower deployment complexity. For example, CoFence [8] can create VNFs upon demand to filter attack traffic, however, security aspects of this approach can be easily tampered with to obtain the incentive related to the mitigation service.

The Trusted Computing and Secure Logging approaches have high deployment complexity as strict hardware requirements need to be considered. Logging is by default provided by any mitigation tool and therefore has no deployment complexity, but secure logging requires a trusted platform to ensure that the output of a mitigation action has not been tampered with. These deployment complexities directly translate to poor scalability since a large number of TPM as well as Intel SGX [2] enabled systems would be required to use a trusted computing approach at scale.

The service model metric differentiates individual approaches by correlating them with their respective cloud service models. Table I presents this metric showing NFV and Trusted Computing approaches follow a similar model as Software-as-a-Service (SaaS) since these are individual software packages that provide the proof. In contrast, secure logging only provides logs identical to Platform-as-a-Service (PaaS) cloud models where an interface to a service is provided. The approach with the highest degree of access to the mitigation system is the network slicing approach where, similar to Infrastructure-as-a-Service (IaaS), a complete virtualized networking infrastructure is provided.

At the outset, it may appear that combining some of these approaches could lead to a comprehensive solution that addresses all requirements. However, as noted in Section III, there exists overlap between most of these approaches, which leads to an increase in complexity and raises scalability challenges. For instance, an NFV approach that is a low complexity approach could be combined with a high-scalability approach like Secure Logging. However, this would lead to combined drawbacks in regards to practicability which would make the resulting approach excel in regards to security compared with the individual approaches but would render it hard to deploy and scale.

The authors envision two main scenarios for future research toward a practicable MaaS offering: The first scenario entails finding a new approach of providing a mitigation proof that balances the security and practicability requirements outlined in the discussion section well enough to warrant a proof-of-concept implementation. The second strategy would introduce the assumption of a minimal degree of inherent trust among the ASes to eliminate the as of yet insurmountable task of isolating the entire mitigation infrastructure in order to prove successful mitigation. Both

strategies could quickly result in a technical demonstration of a complete MaaS architecture to foster further research in the field.

For the first scenario, additional research in the area of trusted computing might turn out to be most fruitful to find a suitable mitigation proof. An approach based on trusted computing that would undoubtedly prove that the entire infrastructure, including the underlying network flows of an AS is tamper-evident, would change the mitigation proof to be an inherent property of the system itself. This would allow for the mitigation to be automatically conducted upon receipt of the attack information while implicitly proving the mitigation service has been carried out correctly due to the trusted infrastructure it is running on.

Existing reputation algorithms can be incorporated into the collaborative defense protocol to foster cooperation and build a foundation of trust in the second scenario. Although being a more viable approach by eliminating the need for software/hardware mechanisms to verify the mitigation of an attack, its sole use does not guarantee that a malicious AS will not perform malicious actions to subvert the functioning of the reputation algorithm such as providing negative feedback to ASes that correctly execute mitigation requests or not providing feedback at all.

V. Final Considerations

Ensuring the effectiveness of a cooperative mitigation task is an important step toward the feasibility of incentives to stimulate the cooperative behavior and to cover operational costs of mitigating attacks. In this regard, this paper presents a detailed and conceptual discussion of four main approaches that could be deployed toward a verifiable MaaS. The multi-dimensional requirement analysis suggests that no single approach by itself can fully implement a trustless multi-domain cooperative defense scenario. Each approach has disadvantages concerning security or practicability and none of the approaches are able to tackle the problem of verifying the authenticity of the network flows.

References

[1] Z. Bozakov and P. Papadimitriou, "Autoslice: Automated and Scalable Slicing for Software-Defined Networks," in *Proceedings of the 2012 ACM conference on CoNEXT student workshop*. ACM, 2012, pp. 3–4.

[2] V. Costan and S. Devadas, "Intel SGX Explained." *IACR Cryptology ePrint Archive*, 2016.

[3] A. Haeberlen, P. Kouznetsov, and P. Druschel, "PeerReview: Practical Accountability for Distributed Systems," *ACM SIGOPS operating systems review*, vol. 41, no. 6, pp. 175–188, 2007.

[4] NFV White Paper, "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. Issue 1," Oct. 2012.

[5] H. Nguyen, B. Acharya, R. Ivanov, A. Haeberlen, L. T. Phan, O. Sokolsky, J. Walker, J. Weimer, W. Hanson, and I. Lee, "Cloud-based secure logger for medical devices," in *Connected Health: Applications, Systems and Engineering Technologies (CHASE), 2016 IEEE First International Conference on*. IEEE, 2016, pp. 89–94.

[6] Nokia, "Trusted NFV Systems," Mar. 2018. [Online]. Available: https://onestore.nokia.com/asset/201400/Nokia_Trusted_NFV_Systems_White_Paper_EN.pdf

[7] R. Perez, R. Sailer, L. van Doorn *et al.*, "vTPM: Virtualizing the Trusted Platform Module," in *Proc. 15th Conf. on USENIX Security Symposium*, 2006, pp. 305–320.

[8] B. Rashidi and C. Fung, "CoFence: A Collaborative DDoS Defence Using Network Function Virtualization," in *12th International Conference on Network and Service Management (CNSM 16)*, October 2016.

[9] S. Ravidas, S. Lal, I. Oliver, and L. Hippelainen, "Incorporating Trust in NFV: Addressing the Challenges," in *Innovations in Clouds, Internet and Networks (ICIN), 2017 20th Conference on*. IEEE, 2017, pp. 87–91.

[10] B. Schneier and J. Kelsey, "Cryptographic Support for Secure Logs on Untrusted Machines," in *USENIX Security Symposium*, vol. 98, 1998, pp. 53–62.

[11] M.-W. Shih, M. Kumar, T. Kim, and A. Gavrilovska, "S-NFV: Securing NFV states by using SGX," in *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*. ACM, 2016, pp. 45–48.

[12] S. T. Zargar, J. Joshi, and D. Tipper, "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. pp. 2046–2069, Fourth 2013.

[13] X. Zhou, R. Li, T. Chen, and H. Zhang, "Network Slicing as a Service: Enabling Enterprises' own Software-Defined Cellular Networks," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 146–153, 2016.

Appendix B

Contents of the CD

The contents of the CD are structured as follows:

Abstract contains the abstract in English.

BloSS contains the complete BloSS source code.

Evaluation contains raw evaluation data for the evaluations outlined in Chapter 6 stored as Microsoft Excel spreadsheets and graphs in PDF format. *Evaluation/Helper* contains the *cpu_logger.py* Python script which was used to gather the CPU usage data used for the evaluations in Section 6.3.

Installation Guides contains detailed installation guides to set up a private IPFS network, InfluxDB, Grafana, Ethereum Netstats as well as the solidity compiler on ARM.

Intermediate Presentation.pdf contains the slides for the intermediate presentation.

Masterthesis.pdf contains the complete thesis report.

LaTeX contains the complete thesis report LaTeX source files including all figures in PDF format

Zusfsg contains the abstract in German.