In this thesis we show that it is possible to build Secure Autonomous Response NETworks (SARNETs); networks that are capable of protecting themselves against attacks. The work describes how control software that utilises Software Defined Networking and Network Function Virtualisation can protect the services in the overlay network. We explain how we combine data from different information sources to improve attack classification and we define metrics that can be used to evaluate defences against attacks in single as well as in multi-domain environments. Finally, we show that multi-domain defence orchestration benefits from considering trust between the collaborating domains. The proof-of-concepts and contributions in this thesis open up new research challenges and introduce ideas that can help to build a more secure and reliable Internet.

Ralph Koning received his MSc degree in 2007 from the Universiteit van Amsterdam. After gaining experience as a network engineer, programmer, and researcher in the System and Network Engineering research group, he decided to pursue a PhD degree in 2015.

This thesis is a collection of Ralph's work between 2015 and 2019 under supervision of prof. dr. ir. C.T.A.M de Laat, prof. dr. L.L. Meijer and dr. P Grosso.

AUTOMATING NETWORK SECURITY - RALPH KONING

AUTOMATING NETWORK SECURITY

RALPH KONING

539064-L-sub01-bw-Koning
Processed on: 10-12-2019

This work was financially supported by the Dutch Research Council (NWO) grant CYBSEC.14.003 / 628.001.016, COMMIT WP20.11. The work in Chapter 5 was funded by ESnet, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-05CH11231.

Cover design by: Hibbem.
Thesis template: `classicthesis` by André Miede and Ivo Pletikosić.
Printed and bound by Ipskamp Printing.
ISBN: 978–94–028–1819–2

AUTOMATING NETWORK SECURITY

Academisch Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K.I.J. Maex
ten overstaan van een door het College voor Promoties
ingestelde commissie,
in het openbaar te verdedigen in de Agnietenkapel
op woensdag 5 februari 2020, te 12.00 uur

door

Ralph Koning

geboren te Amstelveen

Promotiecommisie

Promotor:	prof. dr. ir. C.T.A.M. de Laat	Universiteit van Amsterdam
	prof. dr. L.L. Meijer	Universiteit van Amsterdam
Co-promotor:	dr. P. Grosso	Universiteit van Amsterdam
Overige leden:	prof. dr. P.W. Adriaans	Universiteit van Amsterdam
	prof. dr. T.M. van Engers	Universiteit van Amsterdam
	prof. dr. ing. L.H.M. Gommans	Universiteit van Amsterdam
	prof. dr. R.V. van Nieuwpoort	Universiteit van Amsterdam
	prof. dr. ir. A. Pras	Universiteit Twente
	dr. P.W. Żuraniewski	TNO
	dr. I. Baldin	RENCI

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

To my cats Jason, Shadow, and Nibbler.
May they use this strange rectangular shaped object as a
comfortable place to sleep on.

CONTENTS

1

INTRODUCTION

Crime directed to network infrastructures and network protocols is increasing [44]. The economic and societal consequences of such attacks are reaching front pages in the news, leading society to question their trust in the Internet [41, 84, 112]. These attacks result in service disruptions, bribery, and theft, that costs the victim and society considerable amounts of money. Therefore, companies and governments hire IT security specialists, who deal with improving security of the system and defending against attacks on a daily basis.

As attacks increase in frequency and complexity, it is essential for security professionals to be able to focus on the few important cases that cause considerable damage and to focus on discovering and defending against new threats. This focus can be obtained when a system or network is able to defend itself autonomously against the well known attacks, while gathering more knowledge to optimise its strategies based on past experiences.

At the start of this research in 2015, our group, the System and Network Engineering research group at the University of Amsterdam, considered the use of Software Defined Networking (SDN) and Network Function Virtualisation (NFV) in active mitigation of security problems. At that time, we were aware of only a few researchers that were exploring that field [42, 94]. The limited amount of work on the subject of automated security and defence selection justified an exploratory approach that uses proof of concepts to identify the factors involved in building an auto-response system to cyber attacks. Since attacks can cross multiple networks before reaching the victim, we specifically took into account collaborative defence mechanisms with the aim to put the defence perimeter as close as possible to the origins of the attack.

In this thesis we research how a system or network can self-protect against attacks. We define a conceptual framework, Secure Autonomous Response NETworks (SARNET) and describe how to build a system that defends its own infrastructure. The main features are:

1. monitoring the infrastructure using observables derived from organisational policies to create the necessary awareness of disruptions in the infrastructure;
2. defence using novel technologies such as SDN and NFV that increase the ability to adapt the network infrastructure and its behaviour at runtime;
3. an autonomic computing approach that uses control-loops to automatically respond to disruptions and attacks and maintains the security state of the network;
4. support for the creation of multi-domain alliances that can collaboratively defend against attacks.

1.1 SOFTWARE DEFINED NETWORKS AND NETWORK FUNCTION VIRTUALISATION

To automate network security responses, we need a dynamic (network) environment that is software controllable. Current technologies in cloud computing and computer networking can provide this environment.

Cloud computing allows the users to deploy machines, applications and services, on demand, on standardised platforms that are located all over the world. Cloud computing also changed system administration. Instead of having a powerful server with a complex application stack, administrators use many lightweight Virtual Machines (VM) to run a single application per VM. Running a single application in a VM enables administrators to easily start multiple instances of the application, on demand, and scale up to handle large amounts of users [106, 109].

Since 2005, the SNE research group worked on creating scalable and robust overlay networks using cloud technologies that we called "virtual internets" [66]. These virtual internets, that run on top of the Internet, use User Programmable Virtual Networks (UPVNs) for containment, for isolation, and for constructing a network overlay. In [103], Strijkers showed that he could maintain robustness and scalability by using

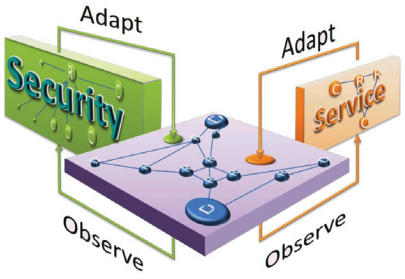


Figure 1.1: A Virtual Internet with with a *service* control-loop that provides scalability and robustness, and the *security* control-loop that we prototype in the SARNET project and is described in this thesis.

CDN providers for attack protection. This method has downsides: using CDNs introduces problems in end-to-end encrypted communication [20], CDN failures cause widespread outages of internet services [21], and CDNs only provide protection if the original connection points are not known [110]. An alternative solution to defend against DDoS attacks is to collaborate with the upstream providers and mitigate the attack in their network, this is the solution we research in this thesis.

1.2 MULTI-DOMAIN SECURITY

Even when having a software-controlled network that is capable of mitigating attacks automatically, defending against attacks can require resources that are not available in a single network. In cases when resources are limited, or unavailable, one can collaborate with other networks that do have the required resources.

On the Internet, each of these networks is referred to as an Autonomous System or a Network domain and each domain is usually operated by a different entity. Therefore, we use the term *domain* according to the definition of *operational domain* as formulated in [31] as: "*A set of network elements with the same operator*". Already, domains have been collaborating with each other on information sharing.

Multiple initiatives exist, either commercial, governmental, or provided by the security community, to share new fingerprints of malware, security incident information, list of misbehaving networks, and attack sources [6, 22, 48]. Other initiatives focus on facilitating collaborative defences. Examples are: DDoS Open Threat Signaling (DOTS), an IETF draft under development to enable multi-domain DDoS mitigation[73], and OASIS OpenC2 who are developing a language for defence command and control that can also be used in multi-domain environments[77].

The existence of multi-domain security initiatives makes it apparent that solving security issues is sometimes a community effort and requires access to resources that are not available within a single domain.

Automated multi-domain defence strategies require that the participating domains can provide the requested information or implement the requested actions within a limited time window. Responding within the limited time window requires the responding domain to have an actively monitored programmable infrastructure.

We use an experimental approach to develop and implement such a system, using proof of concepts. By building proof of concepts, we identify the possibilities and the limitations of such a system and of the used technologies.

1.3 RESEARCH QUESTIONS

The main research question answered in this thesis is:

> *Can we create an auto-response system that defends networks against attacks both within a single autonomous domain and within an alliance of domains?*

Automatic defence requires the detection of abnormal behaviour on the network and on the systems it interconnects. By evaluating the abnormal behaviour, we determine whether or not the network is under attack; this is the security state. To get insight in this evaluation procedure, we formulate the first sub-question:

- RQ1: *How can we determine the security state of a network?*

In Chapter 2 we describe the foundational concepts that are used in the proof-of-concepts which are described in this thesis: the control-loop and observables. The control-loop is used to drive our automated response system which assesses the SARNET's security-state by means of observables. Observables use conditions to indicate abnormal behaviour of the metrics on the network and the connected systems.

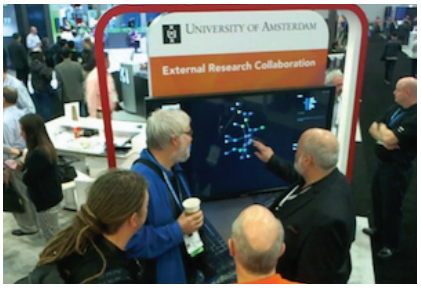


Figure 1.2: At SuperComputing 2015 we demonstrated one of the earlier visualisations at the Ciena booth. The goal was to invite experts in computer networking to defend against attacks using the touch table interface, to engage in discussions, and to capture the knowledge to build automated responses. A detailed view of the screen is provided in Fig. A.2.

Chapter 3 focuses on visualising these observables and metrics such that they can be analysed by engineers using graphical user interfaces that can interact with the network (Appendix A).

The visual analysis process helps security engineers to identify security problems, to develop new software based responses, and to analyse the effect of the response on the security state of the network.

Figure 1.2 shows how we used visualisation to engage in discussions with network experts at the SuperComputing conference to gather the insight to build the automated defences. In Chapter 4 we describe a SARNET-agent and how we use the control-loop to identify and evaluate the security state by monitoring groups of observables. We show how the agent automatically defends the network when the security state is violated.

In Chapter 5 we research how to improve attack analysis and how to obtain a more detailed security state and we study how the analysis process works in a production environment. By developing CoreFlow, a software framework which cross-references data generated by the IDS with Netflow (see Sec. 5.3.2) data collected from routers across the network, we were able to produce new information: the path that the attack traffic took on the network. By defining new observables based on the data produced by CoreFlow, we can classify more precisely to get a more detailed security state.

Techniques in computer networking such as SDN and NFV allow us to change the behaviour of the network. SDNs allow us to reprogram the network on both OSI layer 2 and 3. Using NFV, network functions such as routers, proxies, firewalls, IDS, and honeypots can be strategically placed on commodity hardware throughout the network and can be started anytime when such a function is required. To research how these techniques can be used to construct responses to security threats we formulate the following research question:

- RQ2: *How can new developments in computer network control contribute in creating countermeasures to attacks?*

In Chapter 3 we determine the set of actions that we can perform in a SDN that can be used in defences. In Chapter 4 we describe how we combine the actions from the software defined environment described in Chapter 3 with NFVs that contribute to the security of the network. We describe how we developed multi-stage defences that use NFVs that are encapsulated in Linux containers. For example: to analyse suspicious traffic in more detail we first we start an IDS network function, and use the SDN to redirect the traffic towards this NFV (stage 1). Based on the information gained from additional analysis of the traffic, we can start another containerised function that implements a specific mitigation for the type of traffic (stage 2).

Currently, distributed attacks, such as DDoS, require collaborative defence strategies spanning multiple domains. Collaborating with

domains that specialise in providing security services, or that can help by providing resources, increase the defence capabilities of the defending domain. When defending against an attack, multiple defence strategies can be taken. To construct an efficient defence strategy, it is critical to know which factors play a role when defending collaboratively. Aside from the known factors, such as the cost of placing a countermeasure, and how much information collaborators are willing to disclose, there may be some other factors that play a role in the success of these multi-domain defences, which leads to the final research question:

- RQ3: *Which factors play a role when defending collaboratively, and how do they influence defence efficiency?*

To identify some of these factors, we develop and implement three different approaches of asking collaborators for help (Chapter 6). We compare the performance of these approaches using our the definition of efficiency in Chapter 4, and then we analyse why their efficiencies differ. In Chapter 7 we research a new approach, based on trust, which we compare to the approaches in Chapter 6. The trust-based approach implements the Social Computational Trust Model [27], that uses three components: integrity, competence, and benevolence, to rank the members in the collaboration. We study how a trust-based defence approach can improve collaborative defence efficiency by first requesting resources from the members who are most able and willing to help.

1.4 CONTRIBUTIONS AND THESIS ORGANISATION

Figure 1.3 visualises how this thesis is organised. Chapter 2 introduces the conceptual elements in the SARNET framework, the control-loop, metrics and observables, which we use throughout this thesis. Chapter 2 also explains the steps in the control-loop that are responsible for detection, decision, response, and learning. In Chapter 3 we introduce a VNET, a programmable environment for experimenting with attacks on computer systems. The VNET environment also includes a graphical interface (Appendix A) that has been updated each time we added new functionality to the VNET. It allows a user to analyse and interact with the environment, by initiating attacks or defences. In Chapter 4 we introduce the SARNET-agent that uses the control-loop from Chapter 2 to analyse the network and actuates the developed defences according to the attack classification. We introduce the metrics *impact* and *efficiency*

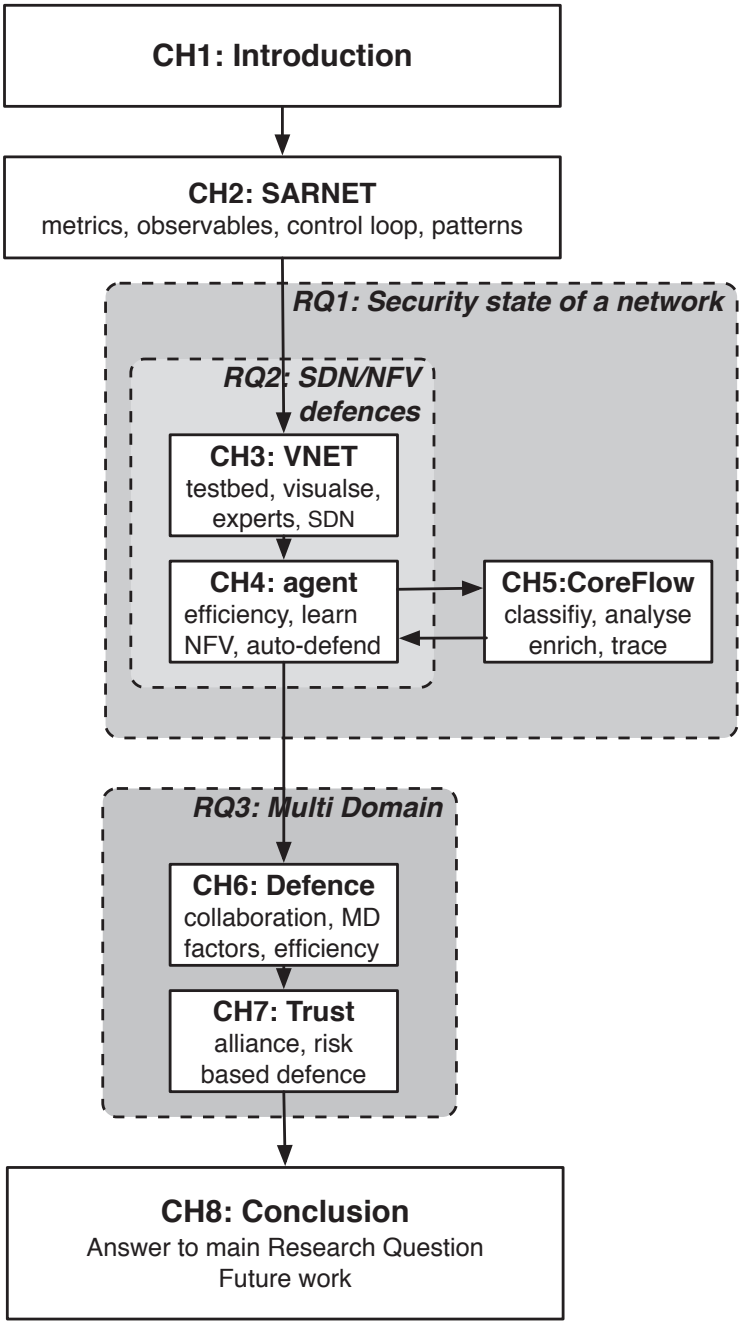


Figure 1.3: Thesis organisation, including the chapters, keywords, and research questions.

that can be used to evaluate defence performance and can be used as discriminators to compare multiple defences against the same attack. The developed defences consist of Network Functions which are dynamically deployed as containers on a cluster, and of instructions for the SDN to redirect traffic to and from the containers. Chapter 5 is a side step into attack-data enrichment. CoreFlow enhances classification by cross-referencing actuations of observables, or other security events, with data from other available information sources. We show how, through this process, we can analyse the cross-referenced data and produce new data that can be useful to apply a targeted defence. Chapter 6 extends the VNET environment to allow for multi-domain experiments where each domain is autonomously controlled by its own SARNET-agent. We show how we allow agents to interoperate with each other by allowing basic informational request and by allowing the execution of basic actionable tasks. We then compare three different approaches which counteract DDoS attacks and analyse how each approach differs in terms of *efficiency*. In Chapter 7 we manage inter-domain trust using the Social Computational Trust Model developed by Deljoo et al. [27]. We extend the SARNET-agent to gather the evidence required for computing trust, and based on that we introduce a trust-based defence approach that we compare against the approaches of Chapter 6.

Finally, in Chapter 8 we answer the research questions RQ1–RQ3 and we reflect on our main research question.

1.5 PUBLICATIONS

A full list of publications by the author is provided at Page 175. The links to the published source code and the datasets can be found at Page 178. Listed below are the author's contributions to the publications that are used in the chapters:

Ch.2 **R. Koning**, A. Deljoo, S. Trajanovski, B. de Graaff, P. Grosso, L. Gommans, T. van Engers, F. Fransen, R. Meijer, R. Wilson, and C. de Laat *"Enabling E-Science Applications with Dynamic Optical Networks: Secure Autonomous Response Networks"* [56], in *Optical Fiber Communication Conference*, © Optical Society of America.
R.K. wrote the operational part, A.D. wrote the strategic and alliances part, and S.T. wrote the tactical part, R.K. consolidated the work and did final editing. The remaining authors edited and supervised the written work.

Ch.3 **R. Koning**, B. de Graaff, C. de Laat, R. Meijer, and P. Grosso *"Interactive analysis of SDN-driven defence against Distributed Denial of Service attacks"* [53], in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, © IEEE.
R.K. worked on developing the Virtual Machine base images, the infrastructure and the defence software prototypes. B.G. designed the visualisation interface and developed the communication layers with ExoGENI with input from R.K. R.K. developed an interface for running experiments and performed the data analysis of the article. P.G. consulted the study. The remaining authors supervised.

Ch.4 **R. Koning**, B. de Graaff, R. Meijer, C. de Laat, and P. Grosso *"Measuring the Effectiveness of SDN Mitigations against Cyber Attacks"* [54], in *2017 IEEE Conference on Network Softwarization (NetSoft)*, © IEEE.
R.K. and B.G. worked on extending VNET; B.G. did most of the programming. R.K. set up and ran the experiments, analysed the data and reflected on the outcomes. P.G. consulted the study. The remaining authors supervised.
R. Koning, B. de Graaff, G. Polevoy, R. Meijer, C. de Laat, and P. Grosso *"Measuring the Efficiency of SDN Mitigations Against Attacks on Computer Infrastructures"* [49], in *Future Generation Computer Systems*, © Elsevier.
R.K. defined the control loop and described its steps. This part has been moved to Chapter 2. R.K. and B.G. worked on extending VNET and implemented the control-loop in the SARNET-agent; B.G. did most of the programming under guidance of R.K. R.K. defined *impact* and worked on a definition of *efficiency*. G.P. later formalised and proved that the definition satisfies the requirements in [81]. R.K. set up and ran the experiments and analysed the data. P.G. consulted the study. The remaining authors supervised.

Ch.5 **R. Koning**, N. Buraglio, C. de Laat, and P. Grosso *"CoreFlow: Enriching Bro security events using network traffic monitoring data"* [52], in *Future Generation Computer Systems*, © Elsevier.
R.K. designed and programmed and evaluated CoreFlow using resources based on input provided by N.B and ESnet. P.G. and N.B contributed by editing the written work. C.L. supervised.

Ch.6 **R. Koning**, G. Polevoy, L. Meijer, C. de Laat, and P. Grosso *"Approaches for Collaborative Security Defences in Multi Network Environments"* [51], in *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th*

IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), ©.
R.K. and B.G. extended VNET and the SARNET-agent for multi domain. R.K. extended the VNET and added multiple defence approaches; G.P. extended efficiency. R.K. did the evaluation of the approaches and the analysis. P.G. consulted the study. The remaining authors supervised.

Ch.7 **R. Koning**, A. Deljoo, L. Meijer, C. de Laat, and P. Grosso *"Trust-based Collaborative Defences in Multi Network Alliances"* [50], in *2019 3rd Cyber Security in Networking Conference (CSNet) (CSNet'19) [accepted]*, © IEEE.
R.K. extended VNET and integrated the trust components. The Social Computational Trust Model is provided by A.D. including the formula for benevolence, competence, and risk. R.K. analysed and evaluated the model in practice. P.G. consulted the study. The remaining authors supervised.
A. Deljoo, **R. Koning**, T. van Engers, L. Gommans, and C. de Laat *"Managing Effective Collaboration in Cyber-security Alliances Using Social Computational Trust"* [26], in *2019 3rd Cyber Security in Networking Conference (CSNet) (CSNet'19) [accepted]*, © IEEE.
A.D worked on the SCTM model and ran the simulations. R.K implemented the model and the algorithm on the VNET testbed and ran experiments. A.D. compared the testbed results with the simulated results. The remaining authors supervised.

2 SECURE AUTONOMOUS RESPONSE NETWORKS

This chapter lays down the conceptual framework for building SARNETs and it highlights the concepts that are implemented in the prototypes and tools used in this thesis. We explain how we approach security problems from a strategic, tactical, and operational viewpoint. We conceptualise the control loop that drives the SARNET-agent in Chapter 4 and describe its steps. We define the concepts metric, observable, task that we use to determine the security state. The security state is used to classify attacks manually, via visualisation, or automatically, by using the SARNET-agent. We also explore how multiple SARNETs can collaborate by sharing information about their security state.

This chapter is based on:

- **R. Koning**, A. Deljoo, S. Trajanovski, B. de Graaff, P. Grosso, L. Gommans, T. van Engers, F. Fransen, R. Meijer, R. Wilson, and C. de Laat *"Enabling E-Science Applications with Dynamic Optical Networks: Secure Autonomous Response Networks"* [56], in *Optical Fiber Communication Conference*, © Optical Society of America.
- **R. Koning**, B. de Graaff, G. Polevoy, R. Meijer, C. de Laat, and P. Grosso *"Measuring the Efficiency of SDN Mitigations Against Attacks on Computer Infrastructures"* [49], in *Future Generation Computer Systems*, © Elsevier.

2.1 INTRODUCTION

The SARNET research goal is to obtain knowledge on how to create ICT systems that model their state, discover by observations and reasoning, if and how an attack is developing, and on how to calculate the associated risks. By collecting state information from systems, network equipment, and databases within a domain we may be able to calculate the effect and the risk of applying possible countermeasures such that we can choose and execute the most suitable one.

In this thesis we recognise that solving computer security problems is not just an operational problem that can be resolved by an engineering team, but requires coordination on all levels within an organisation: strategic, tactical and operational (see Fig. 2.1).

To enable automated response, business values, risks, and policies need to be translated into a set of rules that the software at the tactical level can reason with and can translate into actions for the operational level to execute when such an attack occurs.

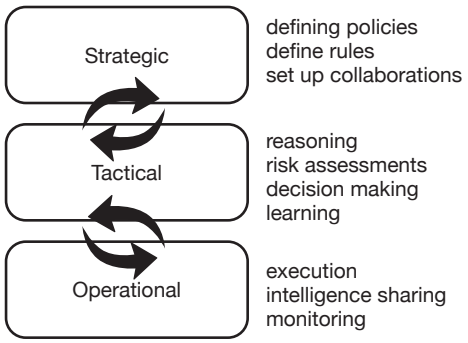


Figure 2.1: SARNETs exhibit coordinated response on three levels: strategic, tactical, operational.

The *strategic* level sets policies, which are expressed as a set of rules to guide and constrain the effective defence strategies against cyber attacks that are operationalised at the tactical level. These strategies can be related to a single organisation or to multiple organisations that collaborate with interconnected SARNETs that we call alliances(Fig. 2.2)

At the *tactical* level, we aim to find response-scenarios that can prevent or mitigate negative effects of an attack on the network. Ideally, the network should autonomously anticipate new threats by taking preventative measures that contain the attack. At the very least, the network should be able to recover efficiently from the attack. Determining the best possible response depends not only on the attack itself but also depends on the environment, cost, and the risks of applying countermeasures. Therefore, it is key to learn from past attacks and to learn from the solutions previously implemented.

The *operational* level provides runtime information to the tactical layer to make its decisions. The runtime information can include an inventory of physical and virtual systems, network topology information, live monitoring information from the network, as well

as the systems' components and security events generated by Intrusion Detection Systems (IDSs). Based on runtime information, the *operational* level classifies suspicious activities on the network and communicates them to the *tactical* level. The *tactical* level then takes this information, makes a decision and picks the most suitable defence from a collection provided by the *operational* level. The *operational* level then decomposes the defences into executable tasks that can be scheduled on and executed by the underlying infrastructure. These tasks need to be executed at the correct place in the infrastructure in order to be effective. Automated scheduling and executing the tasks at the correct place in the network requires the network to be software controllable. We research whether controllability can be achieved by combining cloud facilities in combination with dynamic optical networks, Software Defined Networking (SDN), and Network Function Virtualization (NFV). We choose these technologies as the foundation for creating SARNETs.

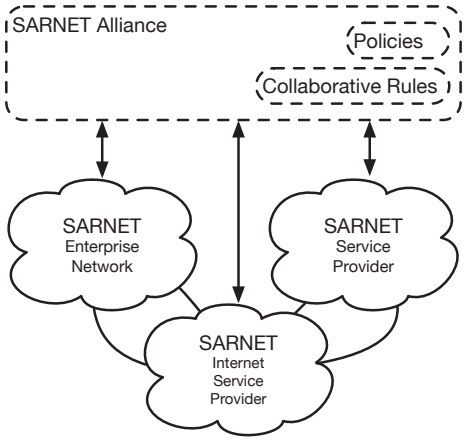


Figure 2.2: A SARNET alliance enforces a common set of rules on collaborating networks.

2.2 THE SARNET FRAMEWORK

Software can efficiently support the instantiation of a network, in any shape or form, as an overlay network on physical devices. SDN, virtual links, and virtual network functions, are the building blocks

for software-defined overlay networks. Organisations increasingly rely on overlay networks for the delivery of digital services to their customers, end-users and other companies. An example is the creation of virtual networks between instances of cloud based virtual machines or containers.

We research if having the capability of re-programming the network and if using software to control the paths taken by traffic flows allows us to selectively block attack traffic or redirect it to (virtual) network appliances for further analysis or mitigation. Automatically providing responses to attacks by reconfiguring the network using software can prevent interruptions and increase the continuity of the services are provided by the SARNET. We develop a framework to increase resilience of a SARNET and the services that it provides by addressing two challenges: Firstly, to enable automated response, the SARNET autonomously reacts to attacks based on metrics that are provided by the SDN and by using a knowledge-base of defences that are tailored to the service itself. Secondly, to increase defence resources, the SARNET needs to allow defences that span multiple domains driven by joint strategies that are executed by the cooperating members.

A SARNET uses control loops to monitor and maintain the desired state required by the security observables. Strijkers et al. [102, 103] researched the idea of using control loops in computer networking. Here control loops were used to enforce connectivity requirements. The requirements were that all nodes in the network needed to be connected to at least two other nodes. The network would automatically recover from link failures and be brought back to the desired state. The SARNET framework captures the idea above and expands on it by describing the elements necessary to implement different and more complex observables and reactions that not only operate on network availability but also on system security.

The SARNET control loop is similar to the OODA loop (observe, orient, decide, and act). Lenders et al. [61] successfully applied the OODA loop to network security. The SARNET loop shown in Fig. 2.3 features more granular steps and explicitly adds a *learn* step to collect data for improved responses to future attacks. Table 2.1 gives a brief overview of the steps in the SARNET control loop, these steps are grouped in four different phases on which we elaborate in the upcoming sections:

- Detection: Detect, Classify, Analyse
- Decision: Risk, Decide

- Response: Respond, Measure, Adjust
- Learning: Learn

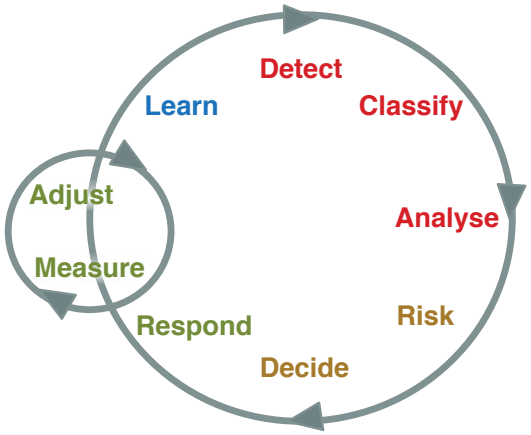


Figure 2.3: The steps and phases in the SARNET control loop: Detection phase (red); Decision phase (orange); Response phase (green); Learning phase (blue).

The control loop and its steps are executed by the SARNET-agent. In *detect*, the agent receives information from one or more external monitoring systems. *respond* relies on software that controls the network elements, a network controller, to delegate the defence tasks to the elements in the network. Together, the SARNET-agent, monitoring system, and network controller, maintain the network's security state.

2.2.1 *Detection: Detect, Classify, Analyse*

Several techniques exist to *detect* known attacks. The first technique is using a signature-based IDSs; these systems can, when updated regularly, detect most known attacks by matching the event to a database of known attack signatures. Flow analysis is another established way of detecting anomalies in the network [97]. Flow analysis can help to detect both known and unknown attacks but requires security experts to identify the anomalies and to collect attack details. Finally, machine learning can be used to detect attacks. Sommer et. al [96] identified five fundamental challenges for the

Step	Description
Detect	Detects the default state in which a SARNET is during normal operation. Whenever the SARNET detects an anomaly on the network it triggers the control loop.
Classify	Classifies the anomaly as an attack and determine what kind of attack it is.
Analyse	Analyses the characteristics of the particular attack. *Analyse* determines where the attacks originate from, which path they take on the network and what the target is.
Risk	Determines the impact of the attack to the business supported by the network.
Decide	Evaluates past decisions and policies and determines the suitable countermeasure for the attack.
React	Executes the countermeasure.
Measure	Samples the new state and check whether the countermeasure is effective.
Adjust	Adjust the reaction where possible to get the optimal result.
Learn	Stores data containing results and execution parameters for future reference.

Table 2.1: The steps in the SARNET control loop

use of machine learning in intrusion detection systems and offer some solutions. Although machine learning is increasingly applied in the field of intrusion detection, some of these challenges, such as obtaining suitable data-sets and identifying suitable features, still hold [104].

The variety of attack detection systems is accounted for in the *classify* step. A SARNET collects metrics from the network, the (intrusion detection) systems connected to the network, and applications. A SARNET is also able to collect metrics based on business processes such as the amount of products that are sold during a time interval. An observable (see: Sec. 2.3) adds a condition to one or more monitored metrics and indicates whether or not the behaviour the metrics is abnormal according to the condition. One or more observables map to a classification.

Based on the classification, *analyse* obtains additional information provided by the network monitoring system about the attack such as: origin, target, entry points, traffic type, and other characteristics. *analyse* also provides information on the scale of the attack which can then be used to calculate the risk of the attack.

To collect this additional information, we describe an event correlation pipeline, CoreFlow (Chapter 5), that queries additional information sources based on events generated by an IDS. The goal of CoreFlow is to extract new data from the combined information. The information generated by CoreFlow can be used for analysis and for more precise attack classification.

2.2.2 *Decision: Risk, Decide*

risk determines the impact of the attack and the chosen defence on the business. The impact can be expressed as revenue loss, yet there are more elaborate models that quantify the impact of an attack, such as [33], that are usable in *risk*. *Decide* evaluates the cost and efficiency of the possible reactions. To make a decision *decide* takes the following aspects into account:

- Attack class (provided by *classify*)
- Attack characteristics (provided by *analyse*)
- Risk of applying the countermeasure (provided by *risk*)
- Knowledge of the network
- Monetary costs of executing responses
- Efficiency of the countermeasure in similar situations (previous results from *learn*)

Effective reaction depends on the characteristics of the SARNET under attack, e.g. whether the SARNET is redundant or multihomed, and it depends on the location where the countermeasures can be applied in the network. In some cases machines or network elements can be added and link capacity can be increased. Dynamically changing link properties is possible thanks to NFV and the cloud services available to the SARNET. A modification will have monetary costs, dependent on the service provider the infrastructure is running on, as well as costs in implementation times, e.g. VM startup times. These costs are parameters that *decide* accounts for. When multiple defences are available for the situation the *decide* step picks the defence with the highest efficiency and executes it.

2.2.3 *Response: Respond, Measure, Adjust*

A defence consists of multiple tasks such as re-configuring the network flows, relocating the virtualised network functions, and relocating computing and storage services. Software Defined Networks (SDN) are an effective way to build and support secure (network) services. SARNET uses SDN to enhance its security by using the SDN's flexibility to change traffic flows and by re-routing important traffic away from overloaded parts of the network towards other parts dedicated to traffic analysis. Combining the flexibility of SDNs with both NFV and machine virtualisation enables deployment of countermeasures where required. Service Function Chaining (SFC), an emerging standard for network control plane operations [83], provides a suitable solution to connect these NFVs together. By using SFC, one can specifically target and re-route suspicious traffic towards network functions that do more intensive processing e.g. DPI, filtering, and scrubbing. DPI is CPU intensive due to the variety of network protocols and data that can be inspected. Dedicated hardware appliances do exist that can do DPI at high speeds, but they are relatively expensive in comparison to regular network equipment. By first separating suspicious flows from the generic traffic, and redirecting the suspicious traffic to such a device, may remove the requirement for dedicated hardware since the amount of data that needs to be processed is lower. Also, since processing is only done on suspicious traffic, delays or interruptions do not affect the regular traffic, which can also relax the requirements. Due to the lower amount of traffic and quality requirements, commodity hardware can be used, which is less costly and more widely available.

Once the countermeasure is activated, *measure* evaluates whether the applied response has the desired effect. If the security observables have returned to normal, the *measure* loop exits, triggering *learn*, and returns back to *detect*.

Defence parameter adjustments may be needed when the network does not recover while defending. This is done in *adjust*. If at a certain point there are no further adjustments possible, or when the adjustments did not provide sufficient improvement in a given amount of time, we will resume the main loop and trigger *learn* that evaluates how well the countermeasure performed and stores this information.

2.2.4 *Learning: Learn*

The *learn* step records the effect of the chosen actions. The data recorded by *learn* allows to respond more quickly to similar attacks in the future. It is essential to properly define the efficiency of a countermeasure. One possible way to express efficiency is using the monetary costs of the response; efficiency is, in this case, the cost recovered thanks to the reaction and cost of the reaction itself. Chapter 4 gives the definition of efficiency and shows how it can be used to evaluate countermeasures. What constitutes an effective countermeasure depends on efficiency but will differ between SARNETs because of variations in network topology, rules, and policies. Therefore, the efficiency is situation dependent. The suitable countermeasures are ranked according to their efficiency and, if the attack characteristics are similar, the ranking can be used in the next iteration of the *decide* phase to choose the best countermeasure.

After *learn* the control loop enters *detect*. In case of a new attack or if the previous attack was not resolved the loop continues again to *decide*. The updated performance information from the *learn* step is incorporated in the decision process and ultimately leads to better decisions and more efficient defences.

2.3 OBSERVABLES

A security observable represents a measurable property of a system. Such an observable changes state when the observed metrics are outside of the desired range or when a certain condition applies. Observables can monitor metrics at the network, application, service and business level. Some examples of observables are listed in table 2.2.

An observable has one or more inputs. The inputs can be dynamic: metrics, observables, or static: test sets, or a fixed value. The output is a Boolean value. Therefore, observable o_i, has two possible states:

$$o_i = \begin{cases} healthy \\ unhealthy \end{cases} \tag{2.1}$$

The output is *unhealthy* when the condition that applies to the observed metric is violated and otherwise it is *healthy*. To map the inputs to the outputs, the observable uses an assigned decision function.

level	**security observable** o_i
business	number of products sold should be above x
policy	traffic to service provider x is passing
services	request to service x is only allowed from y
app	response time of the application should be $<$ 30ms
comp	CPU load of system x should not exceed y
network	Network bandwidth on link x cannot exceed 1 gb/s

Table 2.2: Example security observables defined at different levels.

When an observable's inputs are metrics, the decision function usually contains a ring buffer of x metric samples. This allows the function to detect trends, to count, or to do averaging. The larger the buffer is the slower the (initial) response time of the observable will be. The function can work on aggregated data, for example take the average and compare it to a threshold, or against an average of another metric, or work on individual samples e.g. run a statistical comparison based on the samples in the buffer and the historical data to see how similar they are.

Most of the time, when a single observable triggers, there may not be enough information to classify the event as an attack. Therefore, we allow grouping of specific observables. We group observables by defining a new observable that has the other observables as inputs. Ultimately such a group of observables maps to an attack classification.

Table 2.3 visually shows how observables map to a classification. When the amount of traffic exceeds 80 percent of the link capacity (observable1), the amount of UDP traffic on the link exceeds 80 percent of the total traffic (observable2), and the transactions of a web application go below the threshold of 300 per second (observable3) it triggers the classifying observable, observable4, that maps to an UDP DDoS classification such as described in Chapter 6 Sec. 6.6.

Table 2.3 also shows how the selected defence maps to a set of tasks that need to be executed in a single domain or multi domain environment, depending on the type of defence.

2.4 SARNET OPERATION

Fig. 2.4 shows a simple representation of a SARNET. A simple SARNET has a control loop and security observables (see section 2.2),

Metric	Observable	Classification	Defence	Task
bandwith	>80%	DDoS	Wait it out	start scrubbing
tcp/udp ratio	>10%/90%		Filter locally	redirect clean
transactions	<300		Filter remotely	redirect dirty
			remote scrubbing	

Table 2.3: This table shows how changing observables ultimately maps to defence tasks: when the combination of observables is *unhealthy*, the classifying observable becomes unhealthy and we get a classification (red). Then a defence is selected from a list of defences. This defence consists of multiple tasks (green) that need to be executed or delegated to collaborators in case of multi-domain defences.

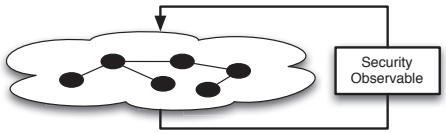


Figure 2.4: A SARNET: the security state of a SARNET i (S_i) is described by the set of the security observables O_i (right) that relate to the desired state of resources (left).

which influence and define the SARNETs security state. The security state is either *OK* when everything is in order, *FAIL* when an attack is classified, and *TRANSIT* when the SARNET is resolving the attack. The security state changes when one of the classifying observables is *unhealthy*. The security state also depends on the control loop state, L. The control loop states are *detect, classify, analyse, risk, decide, react, measure, adjust, learn* and correspond with the steps in the control loop in Fig. 2.3.

The security state and conditions that change the security state are defined as follows:

$$S_i = \begin{cases} OK & \text{if } \forall o_i \in C_i, o_i = healthy \\ FAIL & \text{if } \exists o_i \in C_i \text{ such that} \\ & o_i = unhealthy \text{ and } L = detect \\ TRANSIT & \text{if } \exists o_i \in C_i \text{ such that} \\ & o_i = unhealthy \text{ and } L \neq detect \end{cases}$$

where

S_i = the security state of SARNET i

O_i = the complete set of security observables for SARNET i

C_i = the set of classifying observables for SARNET i; $C_i \subset O_i$

o_i = a single security observable in SARNET i where $o_i \in O_i$

L_i = is the current control loop state of SARNET i

(2.2)

A SARNET can exhibit two types of behaviour: reactive and adaptive: A *Reactive SARNET* consists of an overlay network of interconnected cloud services that maintains a certain security state. If the security state is somehow threatened, the SARNET will adapt the resources in such a way that the threat is minimised and the security state is maintained. An *Adaptive SARNET* constantly changes its network topology using the flexibility of the virtual network infrastructure. The operator defines the input and output requirements of the SARNET. Internally the network constantly changes while still meeting the requirements. A randomly changing network makes it extremely hard for an attacker to discover the true network layout. For example adaptive SARNETs could reduce the attack surface of the network elements by relying on techniques such as IP randomisation [117]. To minimise the negative impact on network performance caused by frequent changes, it is necessary to place measures that negate such influences. Adding appropriate security observables to ensure availability is strongly advised to guarantee continuity of the service running on an adaptive SARNET.

Due to the technical and operational complexities of running an Adaptive SARNET the experiments in the remaining chapters use the reactive SARNET approach.

2.5 IMPLEMENTATION CONSIDERATIONS

When using the SARNET framework to secure networks, it is important to pay attention to network and system design and to take preventive security measures such as network segmentation and applying security best practices:

- Segmentation and separation of concerns:
  A well configured network limits the amount of possible attacks, and a detailed description of this network makes it easier to detect anomalies. Therefore, SARNETs should be bootstrapped in such a way that they have the smallest possible attack surface and they initially enforce a "default deny" approach throughout the whole infrastructure, so that only the approved flows can transit. Communication between application components can be described as policies in a DSL (Domain Specific Language), which are sent to the network controller [59, 86]. The DSL is usually high-level with a focus on communication between application components e.g. only service B can talk to service A. The service meta-data, the network protocols and the ports used in this communication, can be extracted automatically, from the application source code using code analysis, or can be provided by manually describing the service components. When the service meta-data is combined with information specific to the deployment (IP addresses, MAC addresses), the DSL can be used to configure the network elements to only allow these network flows.
- Security best practices:
  The network functions and services that are part of a SARNET should be up to date with the security best practices. For example: routers should implement BCP38[34] and use prefix filtering. To decrease the attack surface, firewalls should be in place for all deployed elements to deny unwanted access to these elements, servers should not run unnecessary services and intermediate switches should drop unnecessary traffic. Vulnerability scanners and software assessment tools can be run periodically to test whether or not the software releases contain known security holes. Periodic assessment limits the vulnerability to any known attacks to the system.
- Protecting the SARNET software components against attacks:
  Also it is important to protect against misuse of the SARNET components themselves. When using software to secure and control the network or the application, this software should be protected against modification by attackers. Common practices

in software engineering such as code signing, anti-tampering protections, and proper access control to the software repositories, are solutions that are also applicable to the SARNET components. These common practices should also be applied to the defence strategies and to the observables.

- Protecting the connected devices:
  Users need to be able to interact with the resources in the SARNET from outside boundaries of the SARNET. One way to provide secure access to the SARNET is providing a web service that can be used to interact with a client side application. The client side application should use encrypted connections, such as Transport Layer Security (TLS) to talk to the web-service or use encrypted Virtual Private Networks (VPN) to connect to the SARNET via the regular internet. Including user devices in a SARNET brings new risks. User devices may contain malicious software and can leak information since they also connect to other networks. Information leakage amongst networks can be addressed by for example, sandboxing the applications on the user device[62] [89]. Unless the user devices are monitored by the organisation and this information is available to the SARNET, the devices are treated as potentially hostile. Extra security measures can be taken to lower the risk of allowing user devices, such as rate limiting the amount of connections, or authenticating the applications.

2.6 INTERCONNECTION PATTERNS

Sometimes its desirable to not use a single SARNET but rather a composition of smaller SARNETs. Sub-dividing a SARNET in smaller SARNETs can help to reduce control complexity, to remove single points of failure, to compartmentalise control and decision making (in case an agent gets compromised), and to reduce response latency and improve decision time. Each SARNET operates autonomously but interacts with its neighbouring SARNET.

We distinguish three patterns for interconnecting SARNETs.

- The Disjoint pattern when there is no relation between the SARNETs;
- The Nested pattern when there is a hierarchical relation between SARNETs;
- The Intersecting pattern when there is a bi-lateral relation between SARNETs.

Besides sub-dividing a single-domain, these patterns can also be used in multi-domain networking: nested SARNETs can be used for a delegated setup, where one bigger organisation or collaboration dictates some of the policies that its members have to comply with. The intersecting SARNET is a useful pattern for bi-lateral peering agreements, where the intersecting SARNET's observables represent the policies agreed upon by the peering parties.

The disjoint SARNET pattern does not impose any requirements on its connected networks, therefore, this pattern is more likely to be adopted. As disjoint SARNETs share no observables with each other and, therefore, operate autonomously, we chose this pattern as a first proof of concept that facilitates collaboration in Chapters 6 and 7. Although the disjoint SARNETs do not communicate their security state, they can collaborate by sending messages to exchange information or requests to execute tasks.

2.6.1 *Disjoint SARNET*

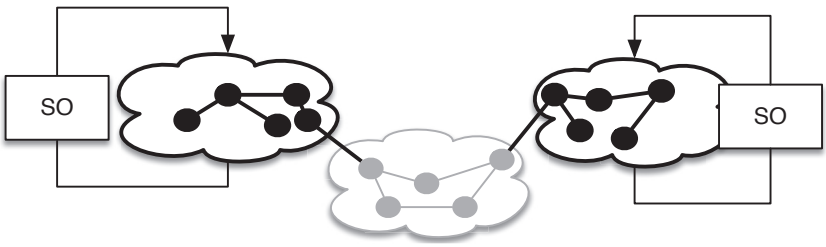


Figure 2.5: Two disjoint SARNETs that collaborate but do not share observables

Disjoint SARNETs (Fig. 2.5) are the simplest form of combining SARNETs. We consider SARNETs disjoint when they both control their own set of equipment and monitor it using their own observables. The connection can however be monitored by either or both of the SARNETs. Disjoint SARNETs have a single requirement: that the demarcation point is controlled by either one of the SARNETs or none at all. An example of a disjoint SARNET is two SARNETs connected using a traditional network. The uncontrolled connection can be a single network link between the SARNETs or traverse several networks that do not implement the SARNET interconnection pattern. In this case the two SARNETs operate autonomously, on

their own observables, and have no knowledge of or visibility into the other SARNET. Because the networks act autonomously and from their own point of view, there is a risk of making conflicting decisions. Decision conflicts can be reduced by adopting more hierarchical patterns such as the nested and intersecting SARNET.

2.6.2 *Nested SARNET*

Nested SARNETs (Fig 2.6) are SARNETs with a dependency, i.e a parent-child relation. The child SARNET acts autonomously and exposes L to its parent.

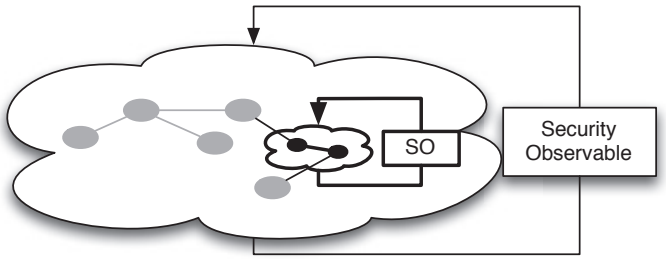


Figure 2.6: A child SARNET is nested in a parent SARNET. The outer SARNET is always aware of the state of the inner SARNET.

A nested SARNET has three types of relationship between child and parent 1) the child inherits security observables from the parent, 2) the child overrides some security observables from the parent and 3) the child inherits none of the observables of the parent. Inheriting is the default behaviour in a Nested SARNET:

1. The child inherits the security observables from the parent and adds its own to extend the observables for a certain part of the network.

$$O_i = O_{parent} \cup O_{child} \tag{2.3}$$

   To avoid race conditions or conflicting actions between parent and child, the child becomes responsible to monitor all observables. Since O_{parent} is exposed to the child, the child acts on

both sets of observables. The security state, S_{child}, is communicated back to the parent. Therefore, the parent knows whether the children are resolving the problem or not, and has the option to respond to that. Also, when the parent observes repetitive changes in the children's security state, or notices that children are interfering with each other, it can solve the issue by taking over the problem solving itself.

2. The child overrides some security observables. It inherits all the observables from the parent

3. the child operates on its own security observables. It inherits non of the observables from the parent and defines its own. Because the child initially has no observables defined or inherited, it can be configured to be less secure than the parent. It is however useful when a part of the network has to operate on a completely different set of observables. When a nested SARNET overrides its parents behaviour there is no relation between its observables:

$$O_{child} \cap O_{parent} = 0 \tag{2.4}$$

also there is no direct relation between the S_{child} and the S_{parent}. The parent is always informed of the current state of its children. When there is no inheritance, race conditions can occur. When $\{S\}_{child} = FAIL$ or $TRANSIT$ the parent should implement observables that watch the child's state and waiting for a certain amount of time to let the child solve its problems and return to OK. It is conceivable that in reaction to the child's change, the parent decides to make changes to the situation in such a way the child changes back to $FAIL$. The interaction can cause parent and child to keep responding to each other and create a feedback loop which can cause the security state of the children to continuously change from $FAIL$ to $TRANSIT$ to OK. The parent should be able to detect these feedback loops and break the cycle by taking charge of the children and enforce another solution.

2.6.3 *Intersecting SARNET*

An intersecting SARNET (Fig. 2.7) connects two SARNETs, without inheriting observables from its neighbours. Defining a separate set of security observables prevents importing conflicting observables

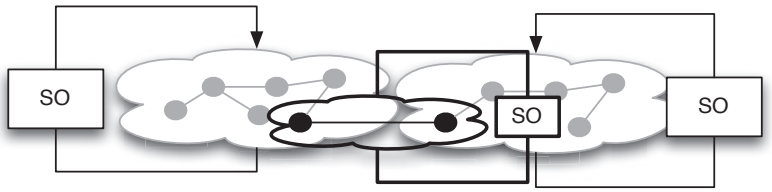


Figure 2.7: A SARNET intersecting two other SARNETs.

from neighbours, that can impact the stability of the network. Separate observables also makes sure policy agreements are reflected correctly. The relationship between security observables can be expressed as:

$$O_i \cap (O_{left} \cup O_{right}) = 0 \tag{2.5}$$

New security observables are defined by the resulting SARNET to enforce the policy agreement, and both parties are aware at of the security state of the intersecting SARNET at any time.

2.7 RELATED WORK

FOCALE [101] is an autonomic networking concept that stands for Foundation, Observation, Comparison, Action and Learning Environment. FOCALE focuses on running network services based on business rules and it provides two control loops, one for maintenance, and one for adjustment of the system components. FOCALE uses semantic data models for reasoning and decision making. FOCALE focuses on providing models running an autonomic network as a whole. SARNET emphasises on security and explores, by developing proof of concepts, whether or not we can actually defend against attacks by applying ideas from autonomic computing to the latest techniques in computer networking.

Internet Factories [103] provide the libraries to orchestrate a network from an application programmer's point of view. An application instantiates an Internet Factory to execute a certain workflow. The Internet Factory builds the network according to the requirements of the application and verifies if these requirements are met. Internet Factories can take care of deploying the application specific network functions in multiple cloud environments. SDN [58] opened up new possibilities to create network operating systems, that monitor and manage the complete network with various levels

of intelligence. The network operating system system provides an API to software developers to create the networks they need for their applications. ONOS (Open Network Operating System) [9], for example, lets the software use something that the authors call "Intents" to describe communication patterns. Another example is ENOS [82], an operating system for the ESnet testbed built by the US Department of Energy, that supports security features and high performance environments. These developments reduce network configuration complexity using higher level abstractions, enable software control and, therefore, enable the deployment of SARNETs.

Security Information and Event management (SIEM) systems [10] collect logging and event data from various components in the network, they correlate them and they generate alerts when suspicious actions take place. It is up to the administrators or security officers on duty to interpret these attacks and respond to them accordingly. SARNET differs from a SIEM by including business policy and business data and implementing automated responses where possible, thus increasing the availability of the service.

There are initiatives that focus on future Internet architectures that enhance security. NEBULA [1], for example, provides secure and resilient networking for cloud architectures. They acknowledge current problems with internet security and solve this by using cryptographic tokens to authenticate packets and a mechanism to verify paths called ICING. However, NEBULA depends on a specific network design and on the use of specific control-plane software. The SARNET framework, although it currently targets SDNs, is technology independent and can also be applied on both traditional and future network technologies.

RINA (Recursive InterNet Architecture) is a complete redesign of the Internet protocol stack currently used and takes away many of the problems that network operators are currently experiencing with using TCP/IP protocol stack [24]. Applications exchange data using one or more DIF (Distributed Inter process call Facility). A DIF remotely resembles a layer in the OSI model [98] and provides a service. The lower DIFs handle packet transmission and routing while the higher DIFs provide application functionality. Each DIF can have its own internal addressing scheme and policies, and to become member of a DIF, the application has to be authenticated, greatly enhancing the security [11].

DRUID (Dynamic Recursive Unified Internet Design) [107] is a future internet design based on the Recursive Network Architecture (RNA), which uses recursive blocks that provide communication services. Unlike RINA, DRUID uses conventional layered architectures

to implement these blocks which can implement security functions to enforce policies and provide the necessary authorisation. The isolation and authentication provided by RNA and RINA seem promising for building secure networks, but the disadvantage is that these architectures requires drastic changes to existing network infrastructure, which limits their adoption. SARNETs use techniques that can be used with existing infrastructure.

2.7.1 *Summary*

This chapter explained the SARNET architectural framework and the core concepts. We explained that we look at the security problem from the strategic, tactical and operational levels. We explained the SARNET control loop and how each step in this loop contributes to maintaining a secure network state. To determine the security state, we introduced the concepts metrics and observables. Finally, we explore patterns to interconnect cooperating SARNETs in both single and multi domain context. The next chapters show our experiences with implementing this framework in practice using both single and multi-domain prototypes that implement the reactive disjoint SARNET pattern. We demonstrate that these prototypes can automatically defend against attacks on the infrastructure under their control.

3

A PLATFORM FOR EXPERIMENTATION WITH SDN-DRIVEN DEFENCES

In this chapter we discuss the testbed that we developed for experimentation with attacks and defences. This chapter explains how we built a software controlled isolated environment for repeated security experiments. We answer RQ2 *by identifying the basic set of response actions that an SDN provides, which can help to defend the network. Using the concepts, metrics, and observables from Chapter 2 we create a visualisation of the network and answer* RQ1 *by demonstrating how these metrics are used for decision making and determining the security state. Finally, we used this visualisation to gain knowledge from experts at the SuperComputing 2015 (SC15) conference, by letting them solve pre-programmed attacks on the network in a limited amount of time. We report on the interactions we had with the experts, and analyse the given solutions, to better understand the strategies that are required to respond automatically.*

This chapter is based on:

- **R. Koning**, B. de Graaff, C. de Laat, R. Meijer, and P. Grosso *"Interactive analysis of SDN-driven defence against Distributed Denial of Service attacks"* [53], in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, © IEEE.

3.1 INTRODUCTION

It is imperative to asses which implementation issues arise when developing a SARNET and use them to improve the architectural framework in Chapter 2. Therefore, we set out to investigate:

- What is an appropriate way to expose security observables to external components, either human or software? Concretely, which visualisation techniques are suitable for SARNETs?
- What is the range of responses possible in a SARNET and how do these responses depend on the underlying SDN control software?
- What are the metrics that can guide the selection of responses to attacks during the *decide* phase, and what are the most valuable metrics we can store in the *learn* phase to determine the solutions' effectiveness for future selection?

In this chapter, we present the results and findings of the above questions that we obtained by letting users interact with our interactive prototype via a multi-touch interface to detect and respond to DDoS attacks. The results were gathered during the Super Computing conference held in Austin, TX in November 2015 (SC15).

3.2 PROTOTYPE ARCHITECTURE

We developed a SARNET prototype called VNET that supports an initial number of SARNET control loop elements, in particular we implement the *Detect*, *Decide*, *Respond*, *Measure*, and *Learn* phases. Currently, VNET is able to provide a visualisation of a network suffering from basic DDoS attacks and it allows users to manipulate the network characteristics with direct visual feedback on their actions and the effects thereof. It allows the creation of simple observables based on the current state of the network topology, traffic and elements. Additionally, VNET allows scripting of attack scenarios, which execute network changes using the network controller. Real-time monitoring data and observable states are forwarded to the UI for visualization and for user interaction. Figure 3.1 shows the application components of the VNET.

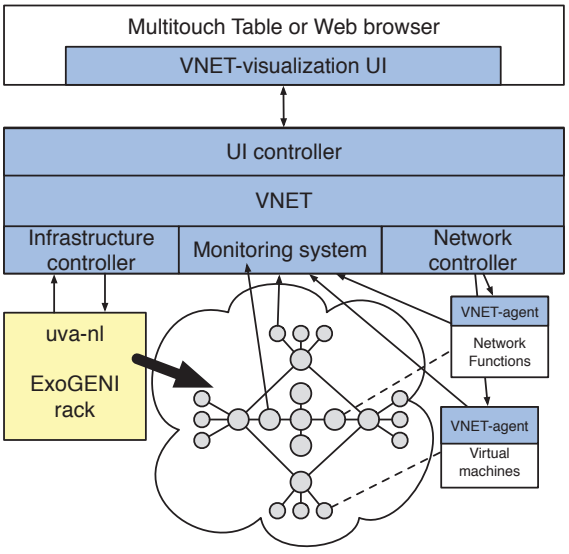


Figure 3.1: Software components in the VNET prototype, version 1

3.2.1 *ExoGENI*

As underlying platform for the VNET operations, we used ExoGENI [5]. ExoGENI is a platform for orchestrating cloud resources (OpenStack), SDNs (OpenFlow)[70], and network circuits[4]. There are currently about 20 ExoGENI racks in operation at various educational and research institutes, including one located at the University of Amsterdam (the *uva-nl rack*). Resources are deployed in the form of slices and can span multiple racks and networks (physical sites). The platform hides the hardware differences between the racks and automatically configures the network. Network elements and functions are implemented as virtual machines.

3.2.2 *Network Functions*

Independence of the underlying infrastructure was an important design requirement. Therefore, we use Ansible[1] playbooks to build our network functions and to prepare the necessary VM images. These playbooks contain instructions to install each virtual machine and to configure the required software packages, including the VNET-agent, which we use to monitor the VM. We currently have network functions for traditional and OpenFlow switches, RIP and OSPF routers, and SDN controllers such as OpenDaylight and Ryu [47]. These network functions are used in the network topologies that are deployed by VNET.

3.2.3 *Infrastructure Controller*

The infrastructure controller acquires and monitors the topology from the underlying infrastructure. This topology consists of the VMs and virtual network links. It translates the topology data from the Infrastructure as a Service (IaaS) controller and converts this to the VNET internal format. The topology is regularly polled at a tunable rate, which can be set to the expected frequency of network changes. A push-based approach can also be used if the IaaS controller supports custom plugins or sends topology updates by itself.

[1] Ansible: https://www.ansible.com/

3.2.4 *Monitoring Controller*

The purpose of the monitoring controller is to collect information, metrics, and statistics from the nodes and links in the network and to pass this to the VNET interface. The node and network state (bandwidth usage and link state per interface) and various function specific data (e.g. spanning-tree information) is sent over an encrypted channel by the VNET-agent that runs on the VMs to the monitoring controller.

In our case, the underlying IaaS platform did not always accurately provide topology information. Thus, we developed the monitoring controller such that it is capable of deducing the topology based on the platform specific meta-data provided by the VNET-agent. For example, in case of ExoGENI, this meta-data contains the URN[2] that uniquely identifies the node, the name of the slice, and the cluster worker node the VM is running on. ExoGENI however, does not provide an URN representation for its interfaces to the VMs. Therefore, the monitoring controller uses the interface IP addresses provided by the VNET-agent to construct the final topology.

3.2.5 *Network Controller*

VNET interacts with the network components and changes their properties to alter the behaviour of the network and the traffic flows. The network controller facilitates this; it uses an RPC channel to the VNET-agents to set node and link configuration, and to execute commands on the nodes. Changes to the network can be triggered by the user via the multi-touch interface or are triggered automatically by VNET.

3.2.6 *VNET-agent*

The VNET-agent monitors and controls the node. The daemon maintains secure connections to both the monitoring controller and the network controller over a TLS secured WebSocket[3] connection. The agent reports detailed interface statistics and host meta-data such as host name, interfaces and IP addresses. Network function specific data can supplement the host meta-data by writing this data in the form of (JSON) key/value pairs to a predefined directory

[2] Uniform Resource Name

[3] The WebSocket protocol: https://tools.ietf.org/html/rfc6455

which will be automatically picked up by the agent and transmitted to the monitoring controller.

3.2.7 *Visualization User Interface*

The VNET user interface supports both touch and pointer events, it is build in JavaScript using the D3.js[4] library and uses WebSockets to talk to the user interface controller component.

Figure A.2 in Appendix A shows the VNET user interface, as it was used during the demonstration at SC15. Information is organised in three columns. The left column shows information relevant to the operation of the network; the demo showed scenario controls and service revenue as described in Sec. 3.3.2. The centre column shows a network representation. It displays disabled links and the line colour changes based on link utilisation for active links. An observable, node health status, is displayed using a red, orange, or green circle. Finally, the right column shows details of the currently selected node or link, and provides controls for node actions.

The concurrent monitoring of both network and service information is an essential element for the SARNETs operation, as this is the only way in which the system can maintain a proper balance between the effect of the executed SDN operations and the services' performance to the end users.

3.2.8 *Bootstrapping*

To bootstrap a network we use a topology description, a JSON[5] file with a list of nodes, node type and a list of all the available network links between these nodes. The format is kept simple for readability and ease of use. This is done by using common defaults, so that properties do not have to be defined for each element. When supported by the underlying infrastructure, multiple domains can be used by simply specifying the location (domain) of the node in the topology. This simplified topology is converted into the appropriate orchestration request for the underlying virtualisation platform. In the case of ExoGENI, this is NDL-OWL [35, 55, 108].

[4] D3.js Data-Driven Documents: https://d3js.org/
[5] JavaScript Object Notation

3.2.9 *Scenarios*

After bootstrapping the network, we load an attack scenario. The scenario definition consists of two parts. The first part defines the initial state of the network such as which links are enabled, what their bandwidth is, and which filters are applied. The scenario also defines the visual elements such as the colours or icons of nodes in the topology. The second part allows for creating predefined attack patterns over a period of time, by defining at what time and by which nodes the DDoS attacks are started. Commands contain the target of the attack, its type (e.g. UDP or TCP), the duration of the attack, and its strength in terms of bandwidth.

3.3 MULTI-TOUCH TABLE DEMONSTRATION

The demo we showed at the Super Computing 2015 (SC15) conference relies on the prototype described in Sec. 3.2. The user of the demo was presented with a multi-touch table interface showing an interactive visualisation of a network. The goal, for the user, was to use this interface to reconfigure the network and minimise the effect of the DDoS attack congesting the service and to recover revenue. By detecting, analysing, deciding and reacting to the attack, the visitor effectively acts as the SARNET control loop (Fig. 2.3). The 25 VMs used for this demo were hosted on the *uva-nl* ExoGENI rack and the links between the nodes were requested with a maximum bandwidth of 100 mbit/s.

During SC15, the ExoGENI IaaS platform posed some limitations: first, running slices could not be modified[6]. We implemented our own mechanisms to scale bandwidth on the virtual links using *tc*[7] on the interfaces of the virtual machines. We used token bucket filter to shape the outgoing rate to the bandwidth requested by the visitor. Additionally, removing links in an active slice was implemented by shutting down interfaces on the virtual machines. Secondly, there was no mapping between the interfaces on the virtual machines and the links in the ExoGENI topology. We solved this by using a static IP addressing scheme that allowed us to unambiguously identify links and interfaces.

We used three types of network elements to build the demo, *routers* running OSPF, *services* and *customers* that can turn into *attackers*. *Services* run a web-service that simulates web-shop transactions;

[6] RENCI deployed slice modification for ExoGENI and deployed this in early 2016.

[7] Tc is a tool to configure Traffic Control in the Linux kernel.

customers send transactions to the web-service while *attackers* send both transactions and attack traffic to the web-services.

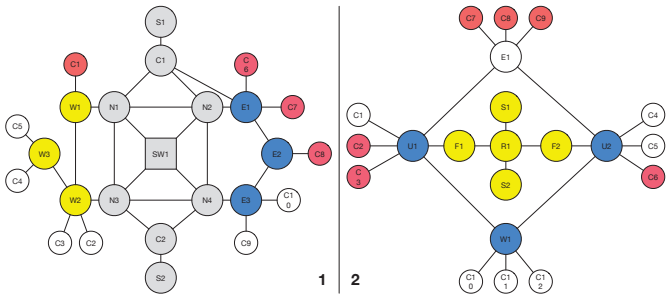


Figure 3.2: The two demo scenarios (scenario 1 on the left and scenario 2 on the right); colours represent different domains

Fig. 3.2 shows the two demo scenarios we used during SC15 where we varied the number of elements present, the network topology among them, and the number of domains present. The network topology is pre-programmed to ensure a correct mix and spread of attack traffic such that the problem is solvable by the visitor and ensure that all the defence strategies have effect.

In both scenarios in Fig. 3.2 virtual customers, C1-12, attempt to perform transactions with two web-services S1-S2. The transactions traverse a network consisting of the routers W1-3, N1-4, E1-3, U1-2 and F1-2. Scenario 1 also includes a switch in the middle, SW1. Some of the virtual customers are assigned the additional role of attacker and try to congest the network such that the virtual customers will be unable to make transactions to the web-services. The dual role of both attacker and consumer is realistic, since attack traffic almost always originates from networks that also send legitimate traffic.

Revenue is determined by taking the sum of the successful transactions between customer and web-service in the network at a moment in time. When the attacks start, the visitor sees that the revenue graph decreases and sees changes in link utilization. Congestion due to attacks causes links to change colour and eventually, since traffic cannot reach the web-services, the web-service icon becomes red as well. This is implemented by using an observable on the amount of sales handled by the service. When the sales drop below a certain threshold, the observable triggers and changes the web-service symbol in the user interface from green to red. When

the service recovers, the visualization turns back to green. The attack traffic consists of UDP *iperf2*[8] traffic from multiple hosts at different rates.

3.3.1 *Responses and costs*

We implemented four responses that can be applied on all links between network elements: 1) link *state*, to shut down links; 2) *rate up*, to scale bandwidth upward; 3) *rate down*, to scale bandwidth downward; 4) *filter*, to filter out attack traffic. Based on the network display, the visitor can apply one or more of these methods on congested links, to restore revenue. The operations have associated costs which are determined as follows:

$$cost = b\frac{\sum_i r_i}{2} + f\sum_i a_i$$

- where b is interface cost in \$ per megabit; we used $b = 10$
- where i is an active interface
- where r_i is link bandwidth of interface i in Mbit/s
- where f is the cost of a filter in \$; we used $f = 500$
- where a_i contains the binary value 1 if a filter is active on interface i otherwise a_i contains 0

The value for b is based on the consideration that bandwidth cost in North-America is \$10 per megabit per second[69]. The value for the parameter f derives from the observation that the market offers DDoS mitigation services from a few hundred to thousands of dollars per month; we therefore choose a relatively high amount(\$500) for a filter action since it is 100% effective when being applied to a link. Since, in this demonstration the number of nodes is fixed, we do not add node costs to the equation.

3.3.2 *Collecting solutions*

To collect defence strategies from all visitors operating the system, we limited the time for each visitor to 4 minutes; during this time, the visitor's goal is to minimise the attack and maximise the sales. Figure A.2 in Appendix A shows the controls for the demonstration on the top-left. Reset stops the attack and starts with a clean scenario.

[8] We used iperf2 instead of the newer iperf3 because iperf2 does not require a control channel for UDP. https://sourceforge.net/projects/iperf/

Start/Retry lets the visitor retry the problem without submitting. *Done/Submit* ends the timer and submits the final result. When the visitor presses *Done*, the revenue over a small time window is measured and this is considered the final score. *Submit* will save the solution.

Per visitor we stored a unique session id and the information listed in Table 3.1 and 3.2.

3.4 RESULTS

The submissions of the users can be used as a dataset to possibly improve automated response to the problems presented in the demonstration. The solutions provided by the visitors in combination with our observations during the demonstration give insight in which attack responses might be effective.

Table 3.1 and 3.2 show the solutions collected during SC15 for the two scenarios described in section 3.3; *rank* shows the best solution, *rank* is currently based on the revenue *recovery* − network *cost*, where *recovery* is the percentage of the revenue that is recovered by the given solution and *cost* is the percentage of the original network cost. *Changes* shows the changes to the topology which is the sum of the actions: link *state* (link down), *rate up* (increasing bandwidth), *rate down* (decreasing bandwidth), and traffic *filter*s.

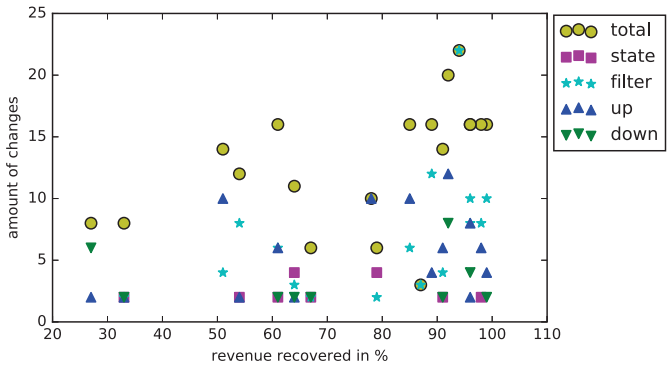


Figure 3.3: Relation between the amount of topology changes and revenue recovery in scenario 1

Fig. 3.3 and Table 3.1 show that filters are used in most of the submissions (16 of 19). The best ranked solution uses no filters but

Table 3.1: Collected data during demonstration: scenario 1

rank	recovery	cost	changes	state	rate up	rate down	filter
1	92	103	20	0	12	8	0
2	91	112	14	2	6	2	4
3	87	108	3	0	0	0	3
4	79	100	6	4	0	0	2
5	98	126	16	2	6	0	8
6	96	127	16	0	2	4	10
7	99	132	16	0	4	2	10
8	67	100	6	2	0	2	2
9	78	114	10	0	10	0	0
10	96	133	16	0	8	0	8
11	64	103	11	4	2	2	3
12	85	129	16	0	10	0	6
13	89	138	16	0	4	0	12
14	61	119	16	2	6	2	6
15	27	94	8	0	2	6	0
16	54	123	12	2	2	0	8
17	94	164	22	0	0	0	22
18	33	103	8	2	2	2	2
19	51	126	14	0	10	0	4

uses only link rate changes, which are less costly. However, the application of filters seem to have a significant effect on revenue. Analysis confirmed the significance of $p = .049$, and the adjusted R^2 of .162 indicates that filters have a positive effect on the revenue.

In Fig. 3.4 and Table 3.2, most of the solutions result in a recovery above 100%. This is possible because of the initial state of the topology where the two links to the web-services were congested by consumer requests. By increasing the bandwidth on these two links, more transactions can reach the web-services resulting in revenue gain. In scenario 2, 15 of the 17 solutions use filters; in contrast to scenario 1, filters did not show a significant effect on the revenue. Analysis of the three similar solutions (rank 2, 3, 4) in 3.2 showed

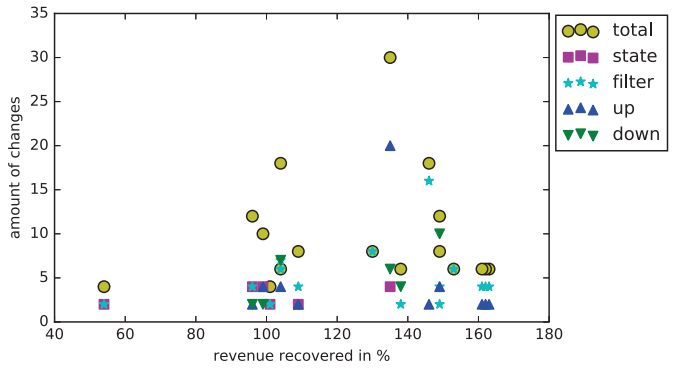


Figure 3.4: Relation between the amount of topology changes and revenue recovery in scenario 2

that they were submitted by different users at different times and the changes were applied to different nodes in the network.

Fig. 3.5 suggests a positive correlation between revenue gain due to the solution and solution costs. Further data analysis showed that the correlation is moderate in scenario 1, $r = .511$, which is significant on a 2-tailed .025 level. For scenario 2, the correlation is weak, $r = .356$, and not significant with a p-value of .161. The figure also shows that most solutions for network 2 provide revenue gain and that it is possible to achieve both revenue gain and network cost reduction simultaneously.

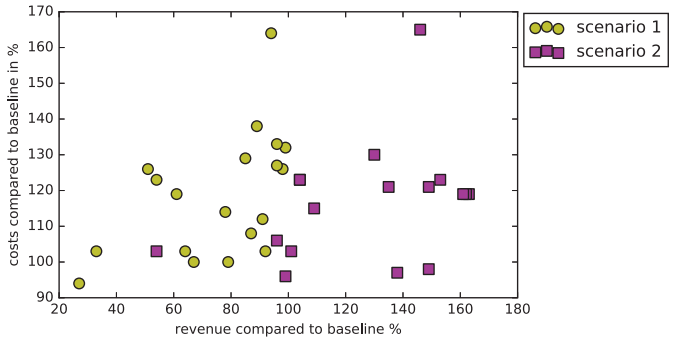


Figure 3.5: Relation between cost increase and revenue increase of a solution

Table 3.2: Collected data during demonstration: scenario 2

rank	recovery	cost	changes	state	rate up	rate down	filter
1	149	98	12	0	0	10	2
2	163	119	6	0	2	0	4
3	162	119	6	0	2	0	4
4	161	119	6	0	2	0	4
5	138	97	6	0	0	4	2
6	153	123	6	0	0	0	6
7	149	121	8	0	4	0	4
8	135	121	30	4	20	6	0
9	99	96	10	4	4	2	0
10	130	130	8	0	0	0	8
11	101	103	4	2	0	0	2
12	109	115	8	2	2	0	4
13	96	106	12	4	2	2	4
14	146	165	18	0	2	0	16
15	104	123	6	0	0	0	6
16	104	123	18	0	4	7	7
17	54	103	4	2	0	0	2

3.5 DISCUSSION

If we look back at the four types of responses that we can perform (see Sec.3.3.1), we can draw general conclusions.

Scaling bandwidth upward or downward requires a careful analysis of the distribution of legitimate network traffic and malicious traffic. In general, upward scaling should be deployed when the attack traffic cannot consume the additional bandwidth, which is then available for legitimate traffic only. This strategy works only when the subsequent downstream links are of equal or higher capacity. On the other hand, scaling downward is the strategy of choice when a large amount of attack traffic traverses a certain link. Scaling down link bandwidth needs to happen as close as possible to the malicious traffic sources to lower the attack throughput that passes on to the rest of the network.

Filtering actions were very effective especially in scenario 1, but also admittedly simple to perform as our attack traffic was always UDP and our legitimate traffic TCP. In case of more sophisticated attacks, defining filter patterns will be more complicated and the execution of complex patterns may require specialized anti DDoS solutions which can be costly especially if one has to filter the higher layers. On the other hand, shutting down links proved to be much less effective. Routing algorithms were, in this case, counter-productive, as they would re-route the attack traffic via other paths in the network, once a link was disabled. These two extremes highlight the need for a proper balance between the traditional routing goal of general availability and the fine-grained behaviour required to fend off attacks.

In scenario 2, we discovered no correlation between types of changes and revenue gain. However, we did saw some successful combinations of countermeasures during our observations when we asked some experienced network engineers to solve the problem. By tactically shutting down links, we were able to reroute most malicious traffic via one path and legitimate traffic over the other; subsequently, by applying bandwidth scaling, up in case of legitimate traffic and down in case of malicious traffic, we were able to restore most of the revenue without applying expensive filtering. Applying these methods results in temporary loss of traffic, but we did not consider this a major drawback as we were not looking for an approach that minimizes impact on the traffic since when under an attack impairment is already the case.

We learned that SDN platforms are suitable for security purposes only if they provide an extensive set of programmable actions. In our case, we used the ExoGENI platform as underlying network infrastructure. At the time, we discovered that the network slices we could create would not exhibit the required flexibility to effectively defend against network attacks. For example, it was not possible to add or remove links or nodes in an existing ExoGENI topology. This implies that the instruction set exposed by the network controllers will limit the response capabilities of the SARNET framework.

Finally, as network topologies increase in size, the choice of the optimal strategy and the decision on where to act will become too complex for a manual solution. Autonomous frameworks such as SARNET will therefore provide the necessary aid to automate the response.

The results showed a positive correlation between the revenue gain and the use of filters for scenario 1 but not in scenario 2. For all the other changes, we found no correlation; therefore, without

further research, we cannot say anything about the effectiveness of the countermeasures because they may be dependent on the specific scenario. Effectiveness of the solution is determined in this demonstration by subtracting the solution cost from the revenue gain which is sufficient for demo purposes. Yet, for real situations other factors may need to be included, e.g. resiliency to network failure.

3.6 RELATED WORK

While security can be enhanced by using SDNs, it is also true that such networks bring in specific attack possibilities. In [91], Scott et al. clearly explain that the features of SDN that are appealing and useful to enhance security are at the same time the ones that can expose these types of networks to novel types of attacks. Sezer et al. [92] compiled an extensive overview of the implementation challenges for adoption of SDN and included a security overview of the possible vulnerabilities. These works teach us that a framework like SARNET will need to include the knowledge of the particularity of the SDN's attack surfaces when compiling the network topology to be instantiated, and limit its exposure to these new attacks. Likewise SDN-specific attacks need to be part of the SARNET knowledge base used in the control loop during the classification phase.

Visualization of network behaviour exists in many prototypes. For example, network weather-maps like [76] are a powerful visualisation method. Still, combinations of monitoring and controlling SDN networks are not yet mainstream and VNET shows how to accomplish both in a simple and intuitive manner. Furthermore, the simultaneous exposure of service and network information is a first step toward a tighter integration of network and applications.

3.7 CONCLUSION AND FUTURE WORK

The SARNET prototype, VNET, and the demonstration at SC15 led to insights into which factors are necessary to autonomously defend against cyber-attacks. We showed that concurrent display of network and service information provide the visual aid required for human analysis of DDoS attacks. Automatic response requires a measure of efficiency, calculating this by subtracting cost from revenue proved to be sufficient for this demonstration. Additionally, the results showed that effective solutions can be achieved using a small number of countermeasures.

In Sec. 3.5 we required a better measure for efficiency in order to evaluate solutions. Chapter 4 provides this measure and demonstrates how it can be used to evaluate responses to attacks. Response actions are limited by the API of the underlying SDN platform, which, in this prototype, prevented us from adding and removing network elements. As IaaS platforms differ in programmability, the range of response actions also differ per platform. In Chapter 7 we introduce the metric *competence* that gives a measure of how well a collaborator can perform a certain task; based on this metric we can see whether executing a defence depending on such an action is feasible or not.

4

MEASURING THE EFFICIENCY OF SDN MITIGATIONS AGAINST ATTACKS ON COMPUTER INFRASTRUCTURES

This chapter presents how we autonomously defend against attacks. We describe how we developed a SARNET-agent that implements the conceptual framework from Chapter 2, running on top of the experimental environment that we described in Chapter 3. To answer RQ1, we demonstrate that we can automatically determine the security state, classify the attack, and deploy a defence. Additionally, we provide the metrics "impact" and "efficiency" to evaluate the performance of the countermeasure. The efficiency of the defence is used to learn which defence to use the next time such an attack occurs. The defences that are used by the agent use the SDN to change the flow of the network traffic and NFVs for analysis and for mitigation of attacks. Our experiences with building these defences gives us the insights necessary to answer RQ2.

This chapter is based on:

- **R. Koning**, B. de Graaff, R. Meijer, C. de Laat, and P. Grosso *"Measuring the Effectiveness of SDN Mitigations against Cyber Attacks"* [54], in *2017 IEEE Conference on Network Softwarization (NetSoft)*, © IEEE.
- **R. Koning**, B. de Graaff, G. Polevoy, R. Meijer, C. de Laat, and P. Grosso *"Measuring the Efficiency of SDN Mitigations Against Attacks on Computer Infrastructures"* [49], in *Future Generation Computer Systems*, © Elsevier.

4.1 INTRODUCTION

A major development in the networking landscape of the past years is the emergence of Software Defined Networks (SDNs). SDNs allow computer networks to be controlled from one or more software controllers using a common interface. These controllers have the ability to monitor and dynamically reconfigure the network, redirect traffic flows and adapt the network to the situation on demand. The question that then arises is whether SDNs can provide novel methods to counteract attacks.

Another emerging technology in computer networking is Network Function Virtualisation (NFV). NFV allows on the fly instan-

tiation and placement of Virtual Network Functions (VNF) in the network [38]. On demand placement of VNFs at the right place in the network and using the SDN to redirect the traffic through the placed VNFs can save resources and their costs. It is immediately clear that NFV has a great potential for network security, especially if we consider that firewalls, IDS, and traffic scrubbing facilities can all be deployed flexibly where most needed.

Our experiences with SDNs in Chapter 3 convinced us that SDNs and VNFs are suitable for attack response mechanisms. In case of attacks on network infrastructure, using SDNs and VNFs has three benefits; 1) detection and countermeasure placement are not tied to the network ingress/egress points but can be anywhere in the network; 2) unused network capacity can dynamically be assigned to handle attack traffic for short amounts of time; 3) deploying countermeasures based on demand brings a reduction of resources that can be assigned to other processes, reducing overall cost.

In this chapter we will use our architecture for Secure Autonomous Response Networks (SARNET) [56]. We will show how SDN-based countermeasures can be adopted for the protection of networks and ultimately for guaranteed delivery of services. We argue that the most useful element of our, or for that matter, any other SDN-based network solution, is a proper characterisation of the countermeasures efficiency. In this chapter we will, therefore, lay the foundation for a generic manner to define and measure the efficiency of SDN-based mitigations against attacks on computer infrastructures. The impact and efficiency metrics presented in this chapter can be used as features in Artificial Intelligence (AI) approaches to improve autonomous response against attacks and to coordinate the actions of VNFs and SDNs, ultimately without external intervention.

4.2 TOWARDS AN ESTIMATE OF EFFECTIVENESS

Given a system like SARNET which uses control loop mechanisms to counter attacks, the interesting part is to determine the effectiveness of countermeasures. We argue that effectiveness needs to be evaluated as a complex value, which has dependencies on the *footprint* of the attack, as well as on the timing of the response. With the former we mean that even if an attack has a specific signature and is recognised as belonging to a known type, the effectiveness of the countermeasure will depend on the specific characteristics of the

attack. When focusing on the time dimension of a countermeasure we define three main intervals:

- Time to detect: t_d
  The *time to detect* is the time from the moment the attack starts (t_{sa}) until the moment the attack is detected (t_{thr-up}), which is the time when the service metrics threshold is crossed. $t_d = t_{thr-up} - t_{sa}$.
- Time to implement: t_i
  The *time to implement* is the time elapsed from the moment the attack is detected until the moment the implementation of the countermeasure is completed ($t_{cm-impl}$).
  $t_i = t_{cm-impl} - t_{thr-up}$.
- Time to recover: t_r
  The *time to recover* is the time elapsed from the moment the countermeasure is implemented to the moment until the service metrics are recovered, and the threshold is passed in the other direction ($t_{thr-down}$).
  $t_r = t_{thr-down} - t_{cm_impl}$.

In terms of the control loop, t_d is the time it takes in the *Detect* phase from the moment there is a trigger to the moment the control loop moves to the next phase. t_i is the time that the control loop spends in the *Analyse* and the *Decide* phases plus the time spent in the *Respond* phase until the moment the countermeasure is in effect. Finally t_r is the time spent in the *Respond* phase until the moment the attack is stopped or mitigated.

Once more, the effectiveness considerations are not just relevant for our SARNET architecture; the results are generalisable in other SDN-based systems. They, in essence, can provide the basis for a standardised and agreed upon set of metrics when comparing different SDN-based response systems.

4.3 THE SARNET PROTOTYPE

To perform our evaluation of SARNETs, we further developed our VNET environment. VNET provides an orchestration and visualisation system for a SARNET; it displays network topology information, network flows and application metrics in an intuitive way. Additionally, it allows the creation of observables based on the current state of the network.

The main components in the VNET environment are depicted in Fig. 4.1. A detailed description of the components is provided in Chapter 3, Sec. 3.2. Here we summarise:

- The *infrastructure controller* talks to the IaaS platform to instantiate the virtual infrastructure; in our case, we use ExoGENI [5] a cloud platform that provides network level isolation.
- The *monitoring system* receives monitoring information from the virtual infrastructure.
- The *network controller* controls the network and hosts in the virtual infrastructure.
- The *VNET-agent* collects monitoring data on the network elements and sends them to the monitoring system and to the network controller for dynamic configuration of the elements.
- *VNET* coordinates the interaction between the different components.
- The *UI controller* and *VNET visualisation UI* display the network information and handle user interactions with VNET.
- The *SARNET-agent* and *SARNET UI* are new components and are described in sec. 4.3.4 and 4.3.5.

To enable autonomous defence we developed a SARNET-agent (Sec. 4.3.4) that receives real-time monitoring data and observable states from VNET and instructs VNET to alter the virtual network infrastructure when action is required. VNET provides the SARNET-agent with the information and the tools it requires for autonomous network defence.

In order to better evaluate our automated defences and support richer responses we updated the initial VNET prototype. First, we added support for VNFs and introduced the infrastructure elements needed to create VNFs that perform certain countermeasures, namely an SDN switch and an NFV-host.

Secondly, we added support for the processing of network flow information. Network flow information is collected by all network routers and SDN switches in the virtual infrastructure using host-sflow[1] and subsequently sent to the VNET monitoring system.

Finally, we updated and refined the SARNET-agent and the User Interface.

The next sections will describe these new and updated components in more detail.

[1] host-sflow: https://github.com/sflow/host-sflow

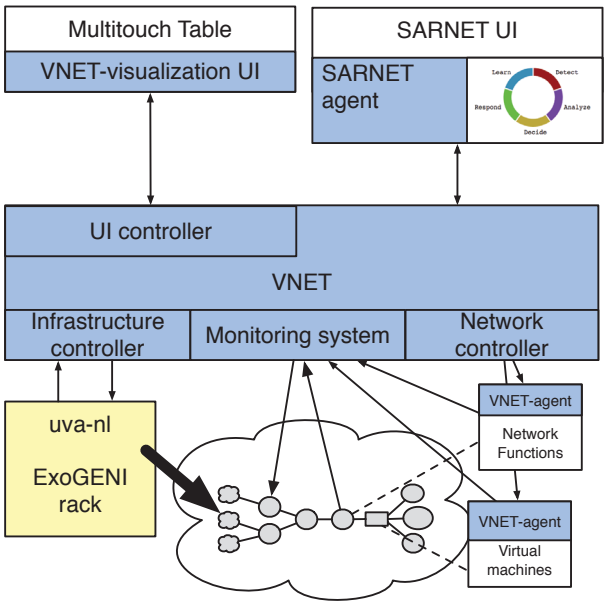


Figure 4.1: Software components in the VNET prototype.

4.3.1 *Containerised Virtual Network Functions*

Three different containers were made to run on the Docker host: an IDS, a CAPTCHA function, and a honeypot.

The IDS container performs packet inspection using PCAP to capture packets. A rule-based engine reports back attacker IP addresses based on known attack signatures.

The CAPTCHA network function acts as a proxy between the external user and the web-service. It will inject a web page containing a mandatory challenge which needs to be solved before the session is allowed through to the web-service it protects. This challenge prevents automated clients from submitting a potentially malicious request. These CAPTCHAs are normally easy to solve by humans but expensive to solve by automated processes. This effectively blocks automated requests, such as attacks, to pass through. Because in this simulation *all* clients are fully automated, we implemented the CAPTCHA defence by using cookies that are only set by non-malicious clients.

The honeypot function simulates a legitimate version of the web-service. However, any interaction with this honeypot will not affect the actual service. The honeypot can be used to capture additional details during an attack. For example, in the case of a password

brute force attack, the honeypot captures the failed password attempts on the attacked account.

4.3.2 *SDN Switch*

The VNET prototype uses software defined networking in order to apply virtual network functions on traffic entering the domain it protects.

The network component that provides the SDN functionality is a Linux host that provides switching through a Linux Ethernet bridge.

In order to redirect traffic flows on this switch, *ebtables*[2] is used to rewrite destination MAC addresses on incoming packets. For example, the destination MAC address on all traffic coming from the switch interface connected to the local router can be rewritten to be destined for a VNF, cluster, or host, for further processing. After processing, the packets can be returned to the switch with the original destination MAC address restored. This results in 'external' packets being redirected *through* the NFV-host, while leaving all other local area network communication unmodified. Selectively redirecting packets, reduces processing overhead on the NFV.

4.3.3 *Network Function Virtualisation host*

NFV allows VNET to deploy specific security functions on traffic flows as needed. The network function virtualisation host is currently implemented as a Linux host with a number of Docker[3] containers. Each container implements a specific network function. A Docker Registry instance is used to store a catalogue of container images.

All containers on the NFV-host are attached to a Linux bridge. By using ebtables traffic to rewrite the destination MAC address, traffic can be forced into a specific container. By redirecting traffic leaving a container towards a next container, various network functions can be chained together. This chaining can be limited to specific IP addresses or IP ranges, allowing only specific traffic to be manipulated.

[2] ebtables: http://ebtables.netfilter.org
[3] docker: http://www.docker.io

4.3.4 *SARNET-agent*

The SARNET-agent implements the SARNET control loop described in Sec. 2.2 which, based on the topology and the data streamed from the monitoring controller, can make autonomous decisions on how to best defend the network. This data is gathered during the *detect* phase.

During the *analyse* phase any changes in service and network state are processed. For example, service transactions per second, CPU usage, and the number of successful and failed logins are monitored. If any of the predefined thresholds for these values are violated, a flag is raised.

In the next phase a *decision* is made based on the currently active flags and any other additional data (e.g. the presence of certain network flow types, data from an IDS, et cetera). Specific combinations of flags and data indicate certain attack signatures for which a set of predefined solutions can be applied. If there is insufficient information about the attack, e.g. the attacking IP address or origin domains are not known, an IDS can be deployed dynamically to gather this information. In addition to applying new solutions, the decide phase also determines whether currently active solutions need to be retained or removed.

In the final phase, the chosen *response* is applied to the network. Possible responses include introducing traffic filters at cooperating upstream routers to block attack traffic, re-routing traffic to the NFV-host using an SDN switch, and choosing the chain of network functions to apply to the traffic.

4.3.5 *SARNET-agent UI*

To show the state of the SARNET-agent and the information it uses to make its decisions we use a visualisation UI (Fig. A.4 in Appendix A) next to the updated interface that is provided by VNET (Fig. A.3) and described in Sec. 3.2.7. The first column (not shown in the figure) shows network metrics such as network flows and total bandwidth usage. The second column shows application metrics such as CPU usage, transaction rate, and successful versus failed login attempts. The final column shows the control loop itself. Each stage of the control loop is highlighted as it is executed, and any decision or result produced by such a phase is displayed in an information block.

4.4 SIMULATED SCENARIOS

To illustrate the SARNET operation of our prototype we have identified three attack scenarios and executed them in a virtual network.

- UDP DDoS attack.
- CPU utilisation attack.
- Password attack

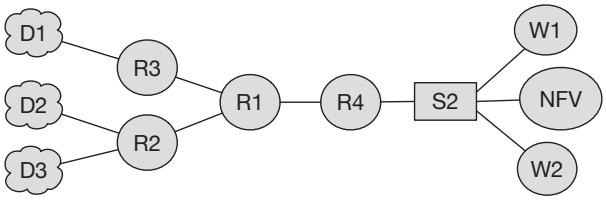


Figure 4.2: Topology of the virtual network: Three domains (*D1–D3*) are connected via multiple routers (*R1–R4*) and a switch (*S2*) to two web-services (*W1–W2*). *NFV* is a host that runs our security VNFs.

Figure 4.2 shows the topology of the virtual network on which we execute the attack scenarios. On the virtual network, traffic passes the virtual routers *R1–R4* and the SDN switch, *S2*, described in the previous section. Together with the web-services *W1–W2* and the *NFV*-cluster, this network is treated as a single domain with a single SARNET agent that monitors and controls the network, which implies that countermeasures can be placed at any node in the network except at the foreign domains *D1–D3*. Under normal circumstances simulated users in the network domains *D1–D3* send regular requests to the web-services *W1–W2* that cause various levels of resource consumption. *D1–D3* also perform both correct and incorrect logins to the web-service using random intervals. The number of successful requests will generate the *sales* value we use in our measurements. In our attack scenarios, attacks originate from the external domains *D1–D3* and target the web-services *W1–W2*.

This virtual network is under constant monitoring. We monitor the following metrics: 1) *sales*, the number of successful transactions to the web-services, 2) *logfail*, the number of failed logins, 3): *cpu*, the CPU load on the web-services, and 4) *traffic_mix*, the ratio between

TCP and UDP traffic on the network. New data for these metrics are collected asynchronously by the SARNET-agent at a sample rate of approximately 1 second. From these metrics we define the following observables that are monitored for health:

- ddos_observable; fails when the metric *sales* falls below its threshold and *traffic_mix* shows excessive UDP traffic.
- bruteforce_observable; fails when the metric *logfail* passes its threshold
- load_observable; fails when metric *cpu* passes its threshold and *sales* falls below its threshold

When one of these observables fails, the SARNET-Agent launches the associated countermeasure.

4.4.1 *UDP DDoS Attack*

In the UDP attack scenario, a number of attackers residing in the same domains (*D1–D3*) as legitimate users send large amounts of UDP traffic toward the servers in order to starve the legitimate connections by congesting the network links. To generate the attacks, we use Iperf2[4] to send non spoofed UDP traffic from all of the domains at a rate specified by the attack size.

The SARNET-agent recognises the type of attack due to the excessive amount of UDP traffic and the simultaneous drop in sales. The SARNET has two possible countermeasures to apply: udp-rateup and udp-filter. In the former we initialize the network link between *R1* and *R4* to 60Mbit and increase the bandwidth of the link using the *tc* traffic control utility to 100Mbit, the maximum available bandwidth and the bandwidth set on the remaining links; in the latter we filter the malicious traffic at the edges (routers *R2–R3*) using *iptables* before the traffic accumulates at *R1* after which the traffic congests the link to *R4*.

4.4.2 *CPU Utilisation Attack*

In the CPU utilisation attack, malicious users in one of the domains *D1–D3* request content from the servers *W1–W2*. Generating content requires computation on the server's side before the request can be satisfied. By requesting computationally expensive pages at a

[4] iperf2 website: https://iperf.fr

high frequency, the attackers increase the CPU utilisation on the servers. This increase, in turn, affects the server's capability to answer legitimate requests. Since these resource requests happen at the application layer, the network layer does not clearly show indication of an attack.

To generate the attack, we change the behaviour of our regular client to CPU attack mode. This mode makes the client malicious by removing delays and by only requesting computationally expensive pages. Attack size depends on the number of attack domains and the number of workers per domain that can be specified, each worker having its own IP address.

In this scenario SARNET first deploys an IDS that performs Deep Packet Inspection in the same domain as the servers to classify and further analyse the requests and to identify attack sources. As the second step, it redirects all requests from the domains where the bad traffic originates, i.e. IP ranges, to a container running a CAPTCHA. The attack requests cannot set the CAPTCHA cookie, preventing the attackers from being proxied to the server. The error returned by the CAPTCHA proxy is computationally cheap, allowing it to handle many more requests than the computationally expensive page on the server. Since the attacks do not pass the proxy, the load on the server returns to normal allowing the server to use its resources for legitimate requests.

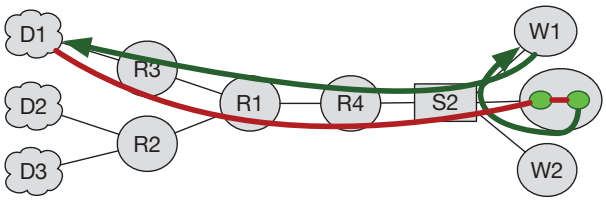


Figure 4.3: The mixed (red) traffic (attack + normal requests) from *D1* is redirected to the NFV-host which has two VNFs chained, first an IDS that monitors the traffic, finally an CAPTCHA blocker that prevents malicious requests to pass and normal traffic (green) to continue to web-services (*W1*–*W2*).

Figure 4.3 shows how the traffic is redirected by *S2* to the NFV-host *NFV* which runs both the IDS and CAPTCHA VNFs. After filling in the CAPTCHA, regular traffic is redirected to the web servers while the automated malicious traffic gets blocked.

4.4.3 *Password Attack*

In the Password Attack scenario, malicious users are trying to log in on the servers using dictionary generated passwords. This attack, as the previous one, takes place at the application layer. It is generated by changing the client to password attack mode. In this mode the client tries to login with incorrect passwords, from a predefined list, without any delays. This results in many incorrect logins. Similar to the CPU attack, the attack size is determined by the amount of attacking domains and the amount of workers per domain.

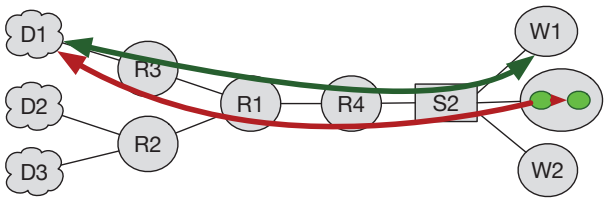


Figure 4.4: The mixed (red) traffic (attack + normal requests) from *D1* is redirected to the NFV-host which has two VNFs chained, first an IDS that monitors the traffic, and a honeypot that can monitor attack behaviour. In this case normal requests (green) pass through untouched to (*W1*–*W2*).

As can be seen in Fig. 4.4, similar to the CPU utilisation attack, the SARNET again responds by first deploying an IDS on the *NFV* host to identify the attackers in *D1*. Additionally, the SARNET starts a honeypot VNF in the container host. The SARNET-agent uses the intelligence information gathered from the IDS to let the SDN switch *S2* only redirect the *identified* malicious users to the honeypot.

Now that the attackers are routed to the honeypot, the web servers *W1*-*W2* can resume normal operations. In principle, the honeypot provides the possibility to further analyse the passwords that the attackers use and to gain additional intelligence. Currently we do not use this to improve the SARNET detection systems; we consider this future work.

4.5 RESULTS

As we have explained in Sec. 4.4, we have considered three types of attacks. In the following paragraphs we will present the results

for the time to detect, time to implement and time to recover in the three scenarios.

4.5.1 *UDP DDoS Results*

A UDP DDoS attack can be described as a function of the injected malicious traffic, resulting in varying degrees of stress on the system. We looked at how our SARNET system responds, as a function of the attack traffic. In our emulation, the three attackers can produce a different rate of UDP traffic, ranging from 20mbps each to a maximum of 80mbps. We apply two responses: filtering on UDP traffic and changing the maximum amount of bandwidth on the link.

When we look at the recovery time of this scenario, we can see that the type of software-defined response we apply in the overlay network has an influence. Figure 4.5 presents this time for the two types of responses we had implemented, the increase of the available bandwidth in the core links or the application of filters at the edges of the network close to the attackers. In the first case (rate change) we observe that at a certain point there is no possible recovery, indicated in the figure with the missing boxplot. This means that in this type of solution, efficiency has a strong relation to the attacker footprint. On the other hand, the application of filters provides a speedy recovery and fairly predictable recovery time.

4.5.2 *CPU Attack Results*

In the CPU attack scenario we simulate a varying number of attackers; we start with 3 and we move on to 5, 10 and 15 respectively. The time to detect a CPU attack does not have a dependency on the number of attackers, as can be seen in Fig. 4.6.

The implementation of the countermeasure has two steps: first we deploy an IDS to classify the requests and secondly we redirect all suspicious connections through a container running a CAPTCHA function. The duration of these two steps is also independent of the number of attackers. This is because these steps are purely related to the software execution times and they take on average 1.73 seconds in our set-up.

Differently from the DDoS attack, in this case, there is clear dependency in the recovery time as a function of the number of attackers. Figure 4.7 shows that the recovery time goes from an average of 6.55 seconds for 3 attackers to 23.5 seconds when there

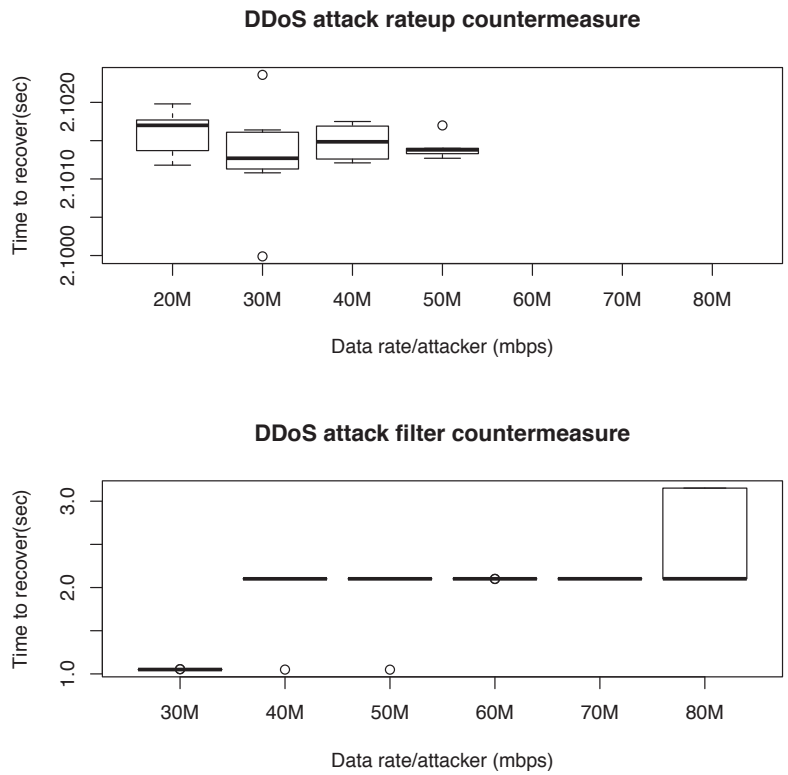


Figure 4.5: Time to recover after the implementation of a countermeasure (in seconds) as a function of each individual attacker UDP rate. Top plot shows the results when applying a rate increase in the core; bottom plot refers to the application of filters

are 15 malicious nodes. This can be explained by the fact that a larger number of attackers brings the amount of successful transactions much further below the threshold, consequently it will take more time to get these transactions back up and pass the threshold again once the countermeasure is in place.

4.5.3 *Password Attack Results*

When we analyse the performance of our system under a password attack we see that the detection time is independent of the number of attackers, as shown in Fig. 4.8. Also, we see that the mean time to detect an attack in this case is lower than the time it took us to detect a CPU attack, namely 1.65 seconds versus 5.26. This

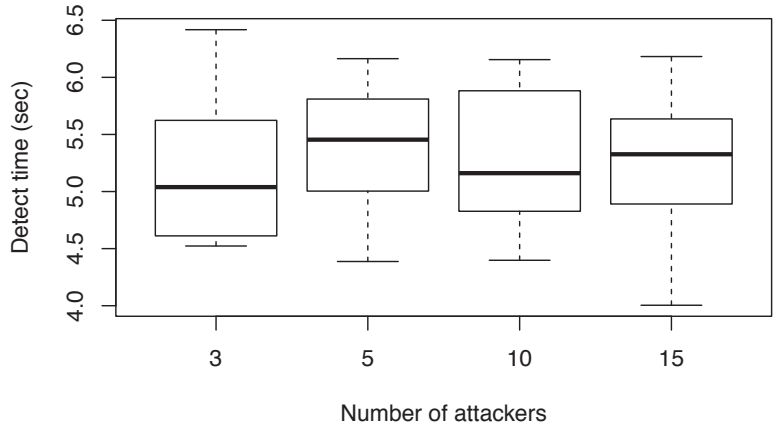


Figure 4.6: Time to detect a CPU attack (in seconds) as function of the number of attackers.

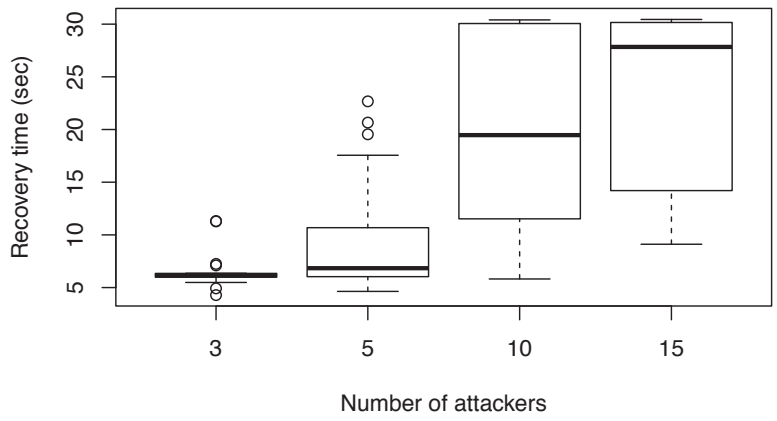


Figure 4.7: Time to recover from a CPU attack (in seconds) as function of the number of attackers.

depends on the way we evaluate the value for the thresholds: a CPU attack requires a separate process that polls the CPU usage on a specified interval while a password attack relies on a counter that continuously updates as failed logins occur.

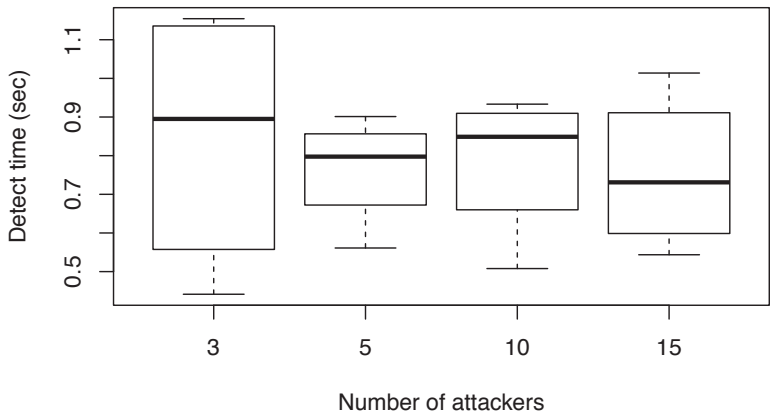


Figure 4.8: Time to detect a password attack (in seconds) as a function of the number of attackers.

The implementation times of the two step defence are shown in Fig. 4.9.

Figure 4.10 shows the amount of time for the system to recover after the successful implementation of the countermeasures in a password attack. In this case there is no dependency on the number of attackers. This is because the redirect to the honeypot happens instantly.

4.5.4 *General Observations*

The three attack scenarios we evaluated show that the detection time and the response time can depend on the attack characteristics, i.e. the number of attackers or the amount of data they transmit. The implementation of a countermeasure in our system is currently constant, because 1) we determined how to react a priori, 2) there is no risk analysis done, and 3) we fully control the devices on which we deploy our countermeasure. The implementation time will start to vary once the risk analysis is more complex and even more so when the implementation steps require coordination with other domains. Latency will increase, thus automatically increase the impact.

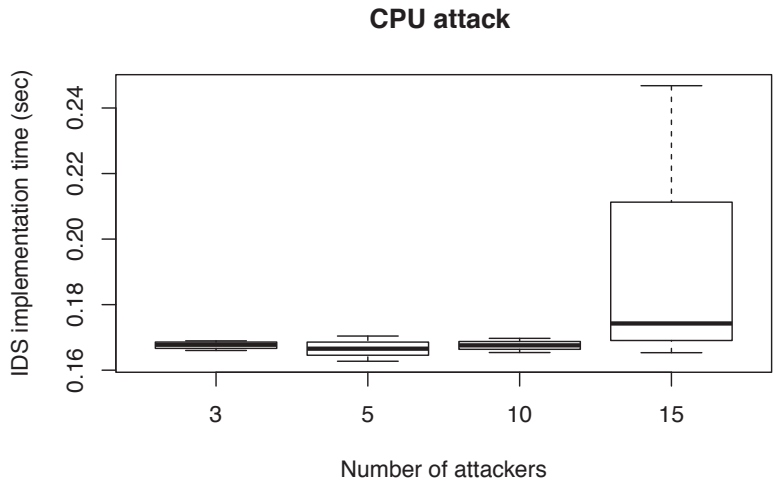


Figure 4.9: Time (in sec) to implement the two step defence in a CPU attack (top) and a password attack (bottom) as function of the number of attackers.

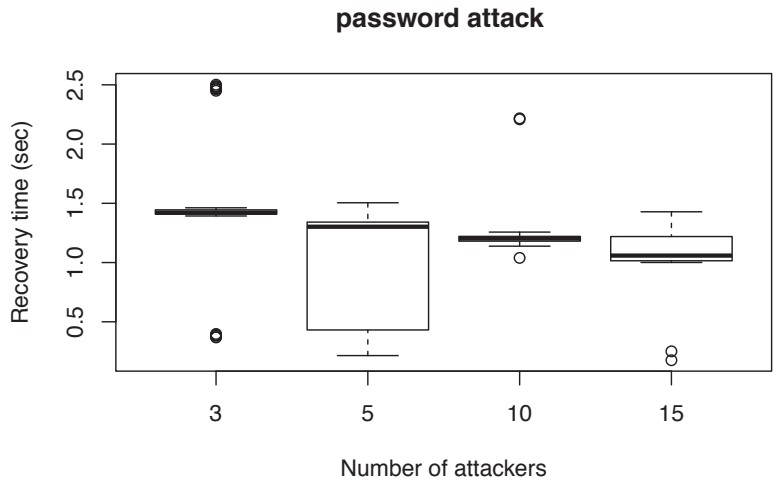


Figure 4.10: Time (in sec) to recover in a password attack as function of the number of attackers.

4.6 TOWARDS AN ESTIMATE OF EFFICIENCY

Given a system like SARNET, determining the efficiency of countermeasures is crucial to estimating how well the system functions

and to learn how to automatically apply the best response. Prior to formally defining efficiency, we define a system for determining recovery and the impact of an attack.

In any given SARNET there will be one or more observables that allow assessing the state of the system: normal or attacked. Each observable monitors a metric in the system and signals a performance degradation when one or more metrics cross a threshold, which could indicate the presence of an attack. The threshold is set according to the outcome of baseline measurements that were performed when the system was under normal operation.

For illustration purposes, we will focus on monetary revenue as our observable, but all our discussion is generalisable to other SARNETs with their relevant observables. For example, if the observable would be the number of failed log in attempts, then being above the threshold would mean an attack, and we would use the same definitions and theory as described below, but adjusted to the new setting.

4.6.1 *Impact*

The impact of an attack can be defined with respect to the chosen observable, such as revenue. First, having set a time window $[0, T]$, we define the system to have recovered if the revenue attains the threshold within the time window. This does not have to occur. It is, in fact, possible that even after the implementation of countermeasures there is no full recovery. In this case, the system achieves a state where the revenue is stable, but still below the threshold.

We now define impact as the integral of the lost revenue between the detection time and the recovery time. If no recovery takes place before the timeout time T, let impact be the integral from the detection time until time T. Fig. 4.11 shows a simplified graphical representation of this concept, when a recovery takes place. Fig. 4.12 illustrates a case without recovery.

The moment at which the threshold is crossed defines the detection time. The revenue may continue decreasing until the countermeasures are in place; then, the revenue starts moving towards the threshold and it either fully recovers, defining the recovery time, or does not fully recover within the time window $[0, T]$. The exact shape of the revenue function during the recovery period depends on the attack characteristics.

In this example, we evaluate the system operations with respect to the revenue and we can calculate the impact by integrating

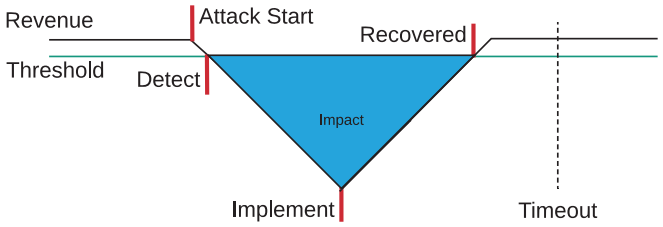


Figure 4.11: Here, a recovery takes place. Impact: the amount of the lost revenue between the detection time and the recovery time (blue area).

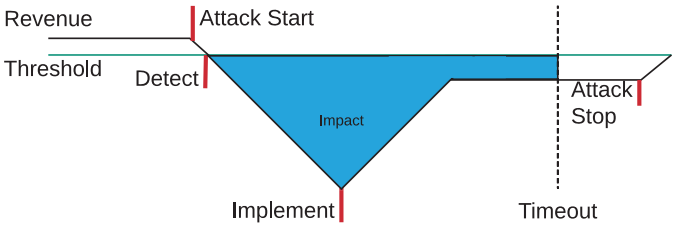


Figure 4.12: No recovery takes place. Impact: the amount of the lost revenue between the detection time and the end of the time window.

the revenue when it is below the threshold. The revenue is lower-bounded at zero, as we cannot have negative revenue; therefore, we do not need to introduce an upper or lower bound. However, there could be cases, in which the observable for which we evaluate the impact can potentially grow/decrease indefinitely, thus requiring the definition of an upper bound. In such cases, we can use an artificial ceiling of twice the threshold, thus setting the scale to be a 100% deviation from the threshold. A side effect of such a ceiling, is that it becomes impossible to distinguish which countermeasure performs better when both the metrics exceed the ceiling i.e. both countermeasures report the value of the ceiling and are considered equally good. Therefore, it is important to let the user adjust the ceiling when necessary.

If no recovery occurs within our time window, one could decide to fine-tune or alter the response, until the recovery is achieved. However, in some cases, the actual recovery is not sufficient to cross the threshold, and thus the system will not fully recover. In these cases we have defined impact as the integral until the end of the time window. Alternatively, we could consider the difference between

the actual recovery and the threshold as a second discriminator in determining how well we recovered.

There are three main elements that affect the impact, as can be seen in the plots in Sec. 4.5.

- the thresholds set to identify attacks will determine the time at which we start to evaluate the integral;
- the scale and characteristics of the attacks themselves might influence the shape of the revenue curve in time;
- the measures that are used to safeguard the network will determine the value of the implementation time and the recovery time.

4.6.2 *Efficiency*

In order to assess the quality of our defences relatively to their total costs and to be able to automatically pick the best defence method, based on past experience, we now define efficiency. The total cost of a defence is defined as the integral of the cost from the attack detection time until full recovery. In case no recovery takes place within this window, we take the integral of the cost until the end of the time window. We emphasise that the definition below and all the theoretical basis for it are fully applicable to any definitions of impact and total cost, as long as the bounds on the values of the impact and the total cost are appropriately defined (they are $B \cdot T$ and $C \cdot T$ in the settings of this section, but can be anything).

We need to define efficiency as a function of whether or not the system has recovered (within the time window), of the impact the attack has had despite our defence and of the total cost of the defence.

Let C be an upper bound on the cost during the period $[0, T]$, and let B be the threshold (or baseline). We require the efficiency function to satisfy at least the following basic properties:

1. Monotonously decreasing with impact I, where $I \in [0, B \cdot T]$. In another setting, $B \cdot T$ should be substituted by the upper bound on I.
2. Monotonously decreasing with total cost Ct, where $Ct \in [0, C \cdot T]$. In another setting, $C \cdot T$ should be substituted by the upper bound on Ct.

3. If no recovery takes place, the efficiency is always smaller than if a recovery does take place, regardless of anything else.

4. All the values between 0 and 1 are obtained, and only they are. In the functional notation, efficiency is a function $E\colon \{\text{recovered, not recovered}\} \times \mathbb{R}_+ \times \mathbb{R}_+ \to [0,1]$.

We define the efficiency as:

$$E(\text{recovered or not}, I, Ct) \stackrel{\Delta}{=}$$
$$\begin{cases} \beta + \alpha\frac{B\cdot T - I}{B\cdot T} + (1-\beta-\alpha)\frac{C\cdot T - Ct}{C\cdot T}, & \text{Recovered} \\ \alpha(\frac{\beta}{1-\beta})\frac{B\cdot T - I}{B\cdot T} + (1-\beta-\alpha)(\frac{\beta}{1-\beta})\frac{C\cdot T - Ct}{C\cdot T}, & \text{otherwise} \end{cases} \tag{4.1}$$

where parameter β defines the cutoff between recovery and no recovery (we allocate β of the total $[0,1]$ scale to the case of no recovery, and the rest is given to the case of recovery), and parameter $\alpha \in [0, 1-\beta]$ expresses the relative importance of the impact w.r.t. the total cost. The idea is to combine the relative saved revenue $\frac{B\cdot T - I}{B\cdot T}$ with the relative saved cost $\frac{C\cdot T - Ct}{C\cdot T}$, and shift the recovered case in front of the non-recovered one. The multiplication by $\frac{\beta}{1-\beta}$ normalises the efficiency of no recovery to fit to $[0,\beta]$.

We now ensure that this function satisfies all the above requirements. The monotonicity in I and in Ct is by definition. The expression $\frac{B\cdot T - I}{B\cdot T}$ can obtain all the values in $[0,1]$, as I is in $[0, BT]$. The expression $\frac{C\cdot T - Ct}{C\cdot T}$ obtains all the values in $[0,1]$, as $Ct \in [0, C\cdot T]$. Therefore, the defined efficiency obtains the values in $[\beta+0, \beta+(1-\beta)] = [\beta, 1]$ if a recovery takes place, and the values in $[0,\beta]$ otherwise. The continuity of the efficiency function implies that all the values in these segments are obtained. To further illustrate the requirements of the efficiency function Eq. (4.1) Polevoy provides a formal proof in [49, 81].

Since in our model we assume that there are no costs for applying countermeasures, we omit the total costs and only consider impacts by setting $\alpha = 1-\beta$. Instead of directly calculating the efficiency of the non-recovered runs, we use the success rates from Table 4.1 to weigh the successful vs. the unsuccessful runs, so we allocate all the range $[0,1]$ for the recovery case by setting $\beta = 0$. After setting $\beta = 0$ and $\alpha = 1-\beta$ we have an equation for the efficiency of a single observable (revenue): $E_m(\text{Recovered}, I) \stackrel{\Delta}{=} 1 - \frac{I}{B\cdot T}$, obtaining values in $[0,1]$.

In order to combine several observables (say, revenues of various kinds), we define the total efficiency as

$$E_{\text{SARNET}} \triangleq \sum_{i=1}^{n} \gamma_i E_{m,i}$$

and we multiply E_{SARNET} by the success rate. Here, the non-negative parameter γ_i describes the importance of ith revenue. By taking γ_is, such that they add up to 1 ($\sum_{i=1}^{n} \gamma_i = 1$), we ensure that E is in $[0, 1]$.

A limitation of defining thresholds and ceilings is that if the thresholds or ceilings are modified over time, the previous values for efficiency and impact have to be recalculated using the new settings to make them comparable to one another. Recalculating efficiency at a later time requires the system to store the complete time series data of each impact for all attacks. Therefore, it is important to choose the threshold and the ceiling carefully before using efficiency in practise.

We have suggested an efficiency function that is characterised by a set of properties, and then we have simplified it for our usage. Therefore, the method for calculating efficiency is not only relevant for our SARNET architecture; the results are generalisable to other SDN-based systems as well. These results can, in essence, provide the basis for a standardised and agreed upon set of metrics when comparing various SDN-based response systems.

4.7 EXPERIMENTAL SETUP

To evaluate whether or not our efficiency definition is suitable to rank the countermeasures applied in the *response* phase, we stop the control loop after implementing the countermeasure and export the data. We refer to each combination of a (predefined) attack and a (predefined) response as a *scenario* and to each execution of such a scenario as a *run*. The experiments are performed on a virtual network in a slice on the uva-nl ExoGENI rack with the topology shown in Fig. 4.2. Each time we start a new scenario, we reset the virtual network to the default state and wait for the network to stabilise.

Sec. 4.4 describes the attack scenarios: DDoS, CPU, and password attack and the four countermeasures we use for the experiments: UDP-filter and udp-rateup, honeypot, CAPTCHA. Note that we consider the deployment of an IDS as a transitory (counter)measure, as it does not provide any resolution to the predefined attacks, but it

only provides extra intelligence information used for a subsequent countermeasure.

In all our runs we define a sample window of 10. We determine that an attack has occurred after more than 30% of the samples of the monitored metrics within the window violate the set threshold. Likewise, we define that the system has recovered when, after the countermeasures have been implemented, more than 70% of the samples within the sliding window pass the predefined threshold in the opposite direction. If there is no recovery within the set amount of time in seconds from detection we time-out and end the run. The ratio of successful runs and failed runs provides the success rate.

Apart from the basic experiment described above, we also define runs where we use a time window of 20, 30 and 40 seconds; these correspond to increase of 2, 3, and 4 times the window size of 10s. To experiment with the success rates, we will allow a relaxation of the recovery threshold. We use recovery threshold relaxations of 0, 5, 10, and 15 percent.

Scenarios are executed 50 times for each combination of attack size, time window size, or threshold relaxation and then we average the times needed for Detection and Recovery; we calculate the Impact following the procedure described in Sec. 4.6; additionally, we calculate the success rate.

We use the results to rank countermeasures, and for each attack-/defence combination we compute the impact and efficiency.

4.8 SIMULATION RESULTS

We will now present the results of running a number of attack/defence scenarios on the SARNET infrastructure.

4.8.1 *Time Evolution of the SARNET*

We can view the behaviour of the system as the time that passes from the start of an attack, the detection and the application of the countermeasures, to the (possible) recovery.

Fig. 4.13 and Fig. 4.14 illustrate two scenarios when we have only one observable governing the state of the SARNET. This is the case in the DDoS scenario and the password attack; in the former the only threshold considered is the revenue, in the latter the threshold is the number of unsuccessful logins.

In both plots the horizontal lines indicate the value of the observable as time passes and the value of the baseline. The vertical lines

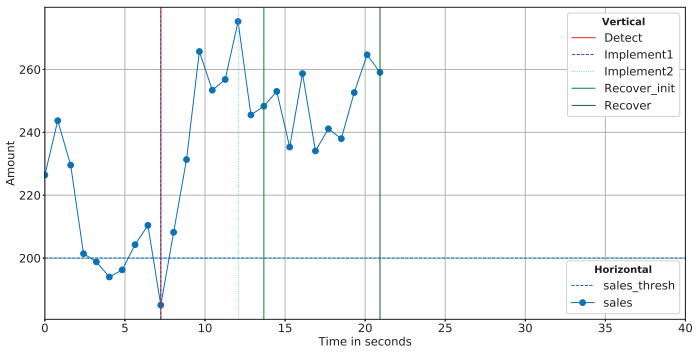


Figure 4.13: Successful run showing the sales metric during a DDoS attack. Note: The implement1 line is plotted on top of the detect line since the events occurred at the same time.

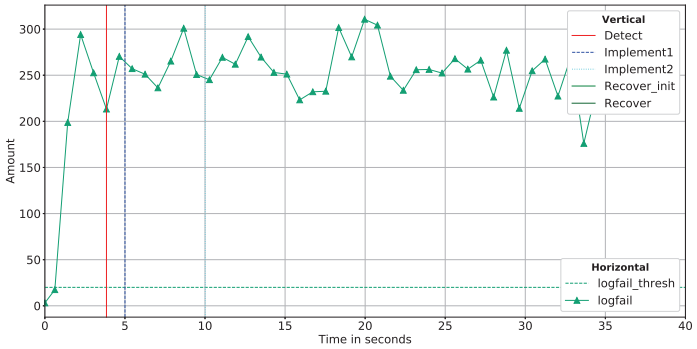


Figure 4.14: Failed run showing the logfail metric during a password attack.

show the detection times, the implementation times implement1 and implement2, and the start and end of the recovery window when the recovery criteria are met. The plots show two different implementation times: implement1 indicates when the agent requests the implementation of a countermeasure and implement2 signals, in case of the *filter* countermeasure the confirmation that the implementation is applied and active. In multi-stage defences, IDS-honeypot or IDS-captcha, implement2 is used to indicate the request time of the second stage (honeypot or captcha).

In Fig. 4.13 a DDoS attack is mitigated and the sales climb back up above the set threshold after the implementation of the countermeasure.

Fig. 4.14 shows an unsuccessful mitigation of a password attack. After recording three samples where the number of logins exceed the threshold, the agent will implement the chosen countermeasure. We see a vertical dotted blue line indicating that the system has implemented the countermeasure but the number of failed logins doesn't fall back below the acceptable value within the allotted time.

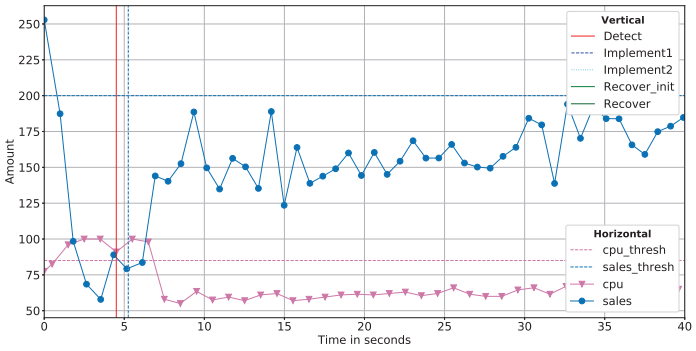


Figure 4.15: Failed defence against a CPU attack where only one metric crosses its threshold and recovers. Note that sales needs to be above threshold and cpu needs to be below.

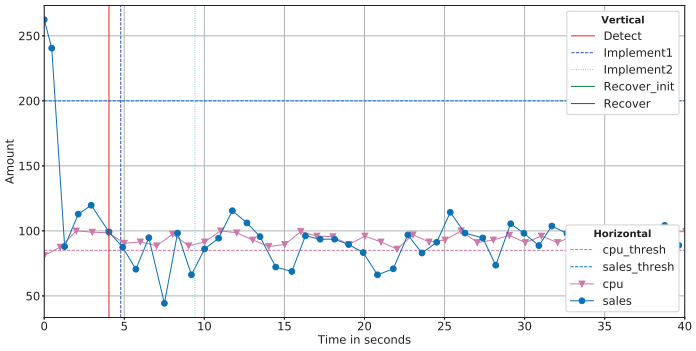


Figure 4.16: Failed defence against a CPU attack where both metrics don't recover.

For the CPU attack recovery needs to happen on multiple thresholds, namely revenue and CPU load. Fig. 4.15 and Fig. 4.16 illustrates two runs where the systems does not recover. In the first case,

the sales do not increase beyond the set threshold while the CPU load does cross its threshold. In the second case, both metrics do not cross their thresholds.

4.8.2 *Success Rate*

We can expect that the success rate of a countermeasure depends on the size of the attack. We distinguish between light, medium or heavy attacks. This characterisation is specific to each attack. For DDoS we define light as being an attack where the throughput of the attackers is 75% of the bottleneck link, medium is 100% and heavy is 200% of the bottleneck link. For both the CPU and the password attacks we define light when attacked by 5 attackers, medium with 10 attackers and heavy with 15 attackers.

Table 4.1: Success ratio of recovery for the various attack intensities as a function of the applied countermeasure.

		% attacks recovered		
	Size	Light	Medium	Heavy
Attack	**Defence**			
cpu	**captcha**	42%	8%	0%
	honeypot	100%	100%	100%
	udp-filter	0%	0%	0%
	udp-rateup	0%	0%	0%
pwd	**captcha**	100%	100%	100%
	honeypot	100%	100%	100%
	udp-filter	0%	0%	0%
	udp-rateup	0%	0%	0%
udp	**captcha**	0%	0%	0%
	honeypot	4%	0%	0%
	udp-filter	93%	100%	100%
	udp-rateup	56%	0%	0%

Table 4.1 lists the success rates of the scenarios. Each variation of the scenario is executed 50 times. An execution is successful when

the metrics cross the threshold and we observe a recovery within the set amount of time, which is 30 seconds by default.

Success rate indicates whether countermeasures are suitable against a specific attack. As we can read from Table 4.1 *captcha* is clearly less effective than a *honeypot* in case of a CPU attack.

However, to further distinguish between successfully recovered runs, we use recovery time. Table 4.2 shows the average recovery time for the scenarios across the same 50 runs as the attack intensity increases. From this table we see that the attack size does not affect the recovery time. There is a 1 second fluctuation which is close to the interval at which we sample the metrics (0.8s).

Table 4.2: Recovery time for successful runs for the various attack intensities as a function of the suitable countermeasures.

		Recovery Time (in seconds)		
	Attack Size	Light	Medium	Heavy
Attack	**Defence**			
cpu	**captcha**	11	10	fail
	honeypot	2	2	3
pwd	**captcha**	2	2	2
	honeypot	2	2	2
udp	**honeypot**	12	fail	fail
	udp-filter	6	6	5
	udp-rateup	10	fail	fail

A system parameter that impacts the success rate of a countermeasure is the time that the system is given to recover before the agent moves on to try the next defence measures. We repeated experiments for three different recovery times 20, 30 and 40 seconds (or 2, 3, and 4 times the window size of 10s) during a Medium sized attack. Table 4.3 shows the success rate of the experiments; as expected the success rate goes up when the time set for recovery is increased.

Many of the failed recoveries are due to the expectation that after application of the countermeasures, the system will return to its original state. As we discussed in Sec. 4.6 there are cases in

Table 4.3: Recovery success ratio for a medium attack with the suitable countermeasures, as the time boundaries are relaxed and the recovery threshold is not relaxed.

		~2x win(20s)	~3x win (30s)	~4x win (40s)
Attack	**Defence**			
cpu	**captcha**	4%	8%	10%
	honeypot	96%	100%	100%
pwd	**captcha**	100%	100%	100%
	honeypot	100%	100%	100%
udp	**udp-filter**	100%	100%	100%

which we can only realistically expect partial recovery. To account for this, we repeated the experiments applying threshold relaxation; we lower the threshold for recovery by a fixed percentage by 5%, 10% and 15%. Table 4.4 shows how the success rate improves as we have relaxed thresholds for various medium attacks; the effect of relaxation is evident in the case of a captcha defence for a CPU attack.

Table 4.4: Recovery success ratio for a medium attack with the suitable countermeasures, as thresholds are relaxed and the recovery time is the set to 20 seconds.

		0%	5%	10%	15%
Attack	**Defence**				
cpu	**captcha**	8%	16%	52%	90%
	honeypot	100%	100%	100%	100%
pwd	**captcha**	100%	100%	100%	100%
	honeypot	100%	100%	100%	100%
udp	**udp-filter**	100%	100%	100%	100%

4.8.3 *Impact and Efficiency*

Sec. 4.6 showed how we determine the impact of the attack and the efficiency of a countermeasure to an attack. Table 4.5 reports on the impact of the attack as a function of the size of the attack. Not surprisingly, we see that the impact of the attack on the system

Table 4.5: Impact of the attack for various attack intensities as a function of the applied countermeasure.

			Impact		
		Attack Size	Light	Medium	Heavy
Attack	**Defence**	**Type**			
cpu	**captcha**	**cpu**	5.64	10.03	fail
	captcha	**sales**	158.19	199.51	fail
	honeypot	**cpu**	5.61	12.14	12.62
	honeypot	**sales**	91.47	119.87	162.57
pwd	**captcha**	**logfail**	22.97	24.17	24.92
	honeypot	**logfail**	24.08	25.57	23.51
udp	**honeypot**	**sales**	41.57	fail	fail
	udp-filter	**sales**	0.26	9.08	25.16
	udp-rateup	**sales**	50.86	fail	fail

increases as the attack size increases. However, in the case of the combination pwd-honeypot, we see a decreased impact when going from a Medium to a Heavy attack. This is due to the artificial ceiling (2x threshold) that we used as a maximum to keep the impact of each measurement within a range. This procedure was described in Sec. 4.5. When we remove this limit, the values for login failures give us the expected increase.

Table 4.6 shows how efficient the countermeasure is in solving the attack; this is the outcome of our efficiency calculation when combined with the success rate. Based on this metric, we can rank the countermeasures, as we did in the last column and we can then use this as the input for the decision phase the next time a similar attack occurs to pick the most optimal solution.

Table 4.6: Efficiency of the countermeasures for the various attack intensities as a function of the applied countermeasure.

		Efficiency			Efficiency × Success Rate			
	Size	L	M	H	L	M	H	
Attack	**Defence**							Rank
cpu	**captcha**	0.98	0.97	0.00	0.56	0.08	0.00	2
	honeypot	0.99	0.99	0.98	0.99	0.99	0.98	1
pwd	**captcha**	0.96	0.96	0.96	0.96	0.96	0.96	1
	honeypot	0.96	0.96	0.96	0.96	0.96	0.96	1
udp	**honeypot**	0.99	0.00	0.00	0.04	0.00	0.00	3[a]
	udp-filter	1.00	1.00	1.00	1.00	1.00	1.00	1
	udp-rateup	0.99	0.00	0.00	0.60	0.00	0.00	2[a]

[a] These rankings are only used in case of Light attacks

4.9 DISCUSSION

Despite the fact that our experiments covered only a limited set of attacks and defences, the method we defined to determine countermeasure efficiency can be universally applied. The only requirement is the availability of time series data on metrics directly associated with the attack class to compute the impact.

We showed that efficiency of the defence depends on the type of attack, therefore comparing the efficiency of different countermeasures only makes sense within the same attack class. Besides attack class there are other factors that influence efficiency:

- the thresholds set to identify attack;
- the time spent on risk analysis and deciding which countermeasure to implement;
- the time allowed for a countermeasure to succeed before going to the next best countermeasure;
- the scale or size and the characteristics of the attacks;
- the execution time of the selected countermeasure.

Because the configuration of the SARNET defines the thresholds and timeouts and because the risk analysis and decisions are common for all countermeasures, the only variables changing during our experiments are the attack scale and the attack characteristics. To get a good measurement for the countermeasure efficiency, the attacks scale and characteristics need to be constrained. In this chapter we used three categories: Light, Medium, and Heavy. Table 4.6 showed that there are indeed different values for efficiency as the scale of the attack changes. For example, the efficiency of a countermeasure during a light attack is not representative when facing a heavy attack. In general, a heavy attack has less effective countermeasures because resource limits (e.g. bandwidth) start to play a role.

In section 4.8 we mentioned that the artificial ceiling that we use to limit excessive values skewed some results. Currently, we made the ceiling dependent on the threshold by limiting the values to a maximum of $2 \times threshold$. Basing the ceiling on the threshold, scales the maximum value together with the expected value of the metric. Normalising the measurement by the maximum amount only gives comparable numbers if the used maximum is equal across all the runs. This limitation also applies to the threshold; one can only compare effectiveness of the runs that use the same thresholds. To compare efficiency in environments with dynamic thresholds, e.g. self learning, or when thresholds are based on time of day, retrieve the raw values for each of the runs and recompute effectiveness each time a comparison is made. Storing the raw data values can consume a considerable amount of storage space.

The countermeasure analysis in this chapter was done after completing all runs. However, the goal is to calculate efficiency each time we defend and update the average defence efficiency immediately. This allows us to update the ranking of countermeasures and pick a more efficient countermeasure the next time such an attack occurs. Eventually, the most efficient solution will be picked first all the time, leaving limited or no experience with subsequent or newly added solutions that do not have an efficiency metric yet. The efficiency metric of a new defence can be gained by forcing new solutions to be tried first. Forcing new and unknown solutions to be picked is not always desirable in a production environment since running unknown, potentially impacting, or sub-optimal solutions can negatively impact the recovery of the service. Limiting time to execute a new countermeasure and immediately following it up with the top ranked defence in case the new countermeasure fails can however be an acceptable strategy. Another approach is

to first test the effects and efficiency of the new countermeasure in a representative staging environment. When the efficiency metrics are collected from the tests in the staging environment, they can be used to update the rankings of the production environment.

Finally, we have to remark that the implementation time per countermeasure is currently constant because the countermeasures are implemented locally. Implementation times will vary more when countermeasures become more sophisticated and rely on information coming in from other sources, or in case of multi-domain defence scenarios when communication times start to play a role such as in Chapter 6.

The results of the DDoS attack Fig. 4.5 show that the rateup defence only recovers below 60Mbit of attack traffic. Since the rate-up can only increase the available bandwidth on link *R1*<–>*R4* (in Fig. 4.2) from 60Mbit to 100Mbit, the maximum bandwidth of the underlying link, congestion still occurs when the attack traffic exceeds the requested extra bandwidth which prevents the victim from recovering. UDP-filter also recovers with larger attack traffic since it takes the attack traffic away at *R3* and *R2* and therefore no congestion occurs at the bottleneck link *R1*<–>*R4*. However, when *R1*–*R3* are in a different network domain and only *R4* is under control of the SARNET-agent, filtering the attack traffic at *R4* will have no effect since the congestion already happened on link *R1*<–>*R4*. For cases like this, where attack traffic already poses a problem before entering the victim domain, we need defence strategies that can work beyond the borders of a single domain.

4.10 RELATED WORK

Our work presents defence mechanisms against cyber-attacks that rely both on SDN mechanism as well as on VNFs in containers. Our ultimate goal is to achieve an autonomous response to such attacks.

Defence mechanisms against attacks have been compared in the literature before. In particular, approaches for the mitigation of DDoS attacks have received significant attention. Surveys have been conducted, for example by Chang et al. [80] or more recently by Zargar et al. [116]. These surveys provide an extensive evaluation of various techniques, but they do not provide quantitative ways to define efficiency as we do in this chapter. Quantitative metrics for efficiency are crucial to support the learning and decision making that is required in an autonomously responding system. Our work, therefore, provides them.

Granadillo et al. [36] describe how countermeasures can be ranked using the RORI index [46] which includes several factors, such as infrastructure costs, risk assessment, and attack surface. This chapter focuses on a subset of the factors considered in RORI. Instead of using an estimative approach, our ranking is based on empirically gathered data on how well countermeasures performed in the past. The way we measure efficiency and impact could be used alongside the RORI model to improve the estimations of future countermeasure performance.

Recent work focuses on the role of SDNs in both providing countermeasures to attacks, as well as identifying unexplored vulnerabilities in SDNs and SDN techniques themselves. Yan et al. [114] address these aspects, and point to the need of extensive evaluation of SDN-based solutions and SDN networks themselves. Our proposal to evaluate countermeasures by efficiency can facilitate future comparison of software based responses.

Our work has shown that some of the components in a counterattack are easily delivered using VNF. In our case, these VNFs are delivered via the deployment of containers at the appropriate locations in the network. Existing work so far has mainly focused on the survey of available techniques and discussing their applicability in various scenarios, particularly in data centres [8] and mobile environments [39] [16]. Previous work has often relied on simulation to assess SDN use as mitigation to attacks, e.g. in the work of Wang et al. [111]. Our application and use of containerised VNFs in a real network that is driven by autonomous responses is, to the best of our knowledge, a first step to show the actual usability and the effect of such techniques.

Autonomy of responses will ultimately rely on machine learning techniques. It has been argued by Sommer and Paxson [96] that machine learning could successfully be applied to the area of intrusion detection. Recent patents such as the one from Google on botnet detection [85] show the applicability of this type approach for identifying attacks. Our ultimate goal of using machine learning to assess efficiency and to adopt the most effective set of countermeasures is, therefore, a novel and promising application of such techniques.

4.11 CONCLUSIONS AND FUTURE WORK

This chapter shows the first steps toward autonomous response to cyber-attacks using SDN and NFV. We showed a first implementa-

tion of the SARNET control loop, from Chapter 2, as a continuation of the VNET work, which after including novel SDN and NFV capabilities, was able to exhibit autonomous response to a selection of attacks.

We introduced a method to compute the impact of an attack and the efficiency of the countermeasure. We evaluated this method by applying it to the attacks and the countermeasures implemented on SARNET and showed how this approach allowed us to rank countermeasures based on efficiency.

Our measurements showed that the detection and the response times are dependent on the attacks' characteristics as well as on the parameters used in the detection and defence systems.

We conclude that metrics for impact of the attack and efficiency of a countermeasure can be applied universally and are valuable inputs in selecting the most suitable countermeasure to an attack.

Some attacks like the UDP DDoS attack can cause congestion on links in networks that are not controllable by the defending domain and prevent the victim from recovering. Defending against such a distributed attack requires coordinated defences across multiple domains. Chapter 6 discusses collaborative defenses between multiple network domains against such attacks.

We also showed that is it possible to develop and deploy countermeasures as containers. We see potential for containers to be used as a method for sharing security VNFs, such as detection mechanisms, or countermeasures, in a reusable manner. In practice, this requires methods to verify that these VNFs cannot negatively impact performance or security of the host network. The challenges around securely executing containers is researched in the DL4LD [32] and SecConNet [95] projects.

5

ENRICHING SECURITY EVENTS WITH IPFIX AND TOPOLOGY INFORMATION

To see which factors play a role in attack classification in a production environment, we collaborated with ESnet to develop CoreFlow. Coreflow is a pipeline for cross-referencing security event data generated by Intrusion Detection Systems with data from other sources. We developed a method which, by cross-referencing security events from the Bro IDS, NetFlow, routing, and topology information, identifies how the attack traffic traverses the network. By knowing the path of the attack traffic, one can narrow down the elements that can play a key role in the defence. CoreFlow is now used as a proof of concept for CEASE [13, 64], the security framework for the upcoming deployment of the US Department of Energy Sciences Network (ESnet). This chapter contributes to answering RQ1*, because the tools and methods described in this chapter provide the attack details and the context to more accurately determine the security state.*

This chapter is based on:

- **R. Koning**, N. Buraglio, C. de Laat, and P. Grosso *"CoreFlow: Enriching Bro security events using network traffic monitoring data"* [52], in *Future Generation Computer Systems*, © Elsevier.

5.1 INTRODUCTION

There are many developments in monitoring and intrusion detection systems (IDS) that enable them to trigger alerts when such activities are present [25, 65]. When such an episode occurs, it is the responsibility of the security and incident response teams who monitor this information to further investigate these events; this often requires them to search information in multiple sources to make a more informed judgment. In this chapter we describe CoreFlow; a prototype framework to enrich IDS data with network flow data. Using CoreFlow to enrich IDS data provides more context to security events. The extra contextual information can then be used to create more targeted alerts and more advanced responses. Automating the cross-referencing process is particularly important for carrier networks that, due to their size and characteristics, require to correlate information from distant elements in the network.

Aspect	Enterprise/Campus	Carrier/Transit
external connectivity	limited (single or redundant uplink)	many connected networks
application security	security can be tailored to application	need to allow everything
restrictions and policies	can be applied anywhere	subject to net neutrality laws
impact of countermeasure	may affect users of a host or system	can affect many users and other networks
network capacity	accommodates one organization	accommodates many institutions

Table 5.1: Major differences between Enterprise/Campus networks and Carrier/Transit networks that are relevant from a security point of view

In section 5.2 we will briefly review the different challenges carrier networks face to secure their networks, and we introduce ESnet, the network in which we tested CoreFlow; in section 5.3 we discuss the information sources used in this research. Section 5.4 and Section 5.5 describe the CoreFlow architecture and implementation. In section 5.6 we reflect on the functionality of the framework and discuss what can be improved. Section 5.7 covers related work and section 5.8 contains the conclusion and future work.

5.2 CARRIER NETWORK SECURITY

Carrier networks present different challenges from enterprise or campus networks, due to their different characteristics. In table 5.1 we list five aspects in which carrier networks differ from enterprise and campus networks when we consider them from a security perspective: external connectivity, application security, restrictions and policies, impact of countermeasures and network capacity. For example, in carrier networks, it is infeasible to run all traffic through a single or a set of security appliance devices due to very high data rates, as well as the large or numerous data flows and multiple ingress and egress points. Additionally, carrier networks are often tasked with adhering to network neutrality laws or policies which prevent filtering or altering traffic in any way, other than to protect the infrastructure of the network.

5.2.1 *ESnet*

Our CoreFlow development and validation has taken place at ESnet. ESnet is a national research and education network (NREN), which, besides providing internet access, interconnects the national labs in the US with research institutions, super computing facilities, and research networks all over the world.

Figure 5.1 shows the topology of the ESnet backbone network that spans the US and a part of Europe. The backbone consists mainly of 100Gbps links and allows sites to connect to ESnet at various speeds.

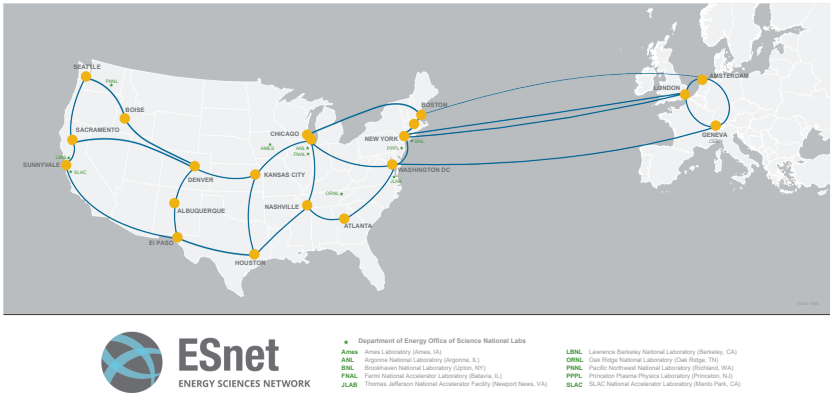


Figure 5.1: ESnet network. Source: http://www.es.net

NRENs such as ESnet are mainly built to *transport* data between the connected institutions and do not act as a network end-point, therefore, they are considered carrier or transit networks. Since NRENs transport large amounts of data, the chances that attacks occur are higher and, therefore, they are a suitable testing ground for CoreFlow.

5.3 INFORMATION SOURCES

Different information sources can be used to identify and counteract network attacks.

IDSs are able to perform in-depth inspection of packets to detect security problems, yet they only have a limited perspective of the

network. NetFlow and other flow-based tools provide detailed network traffic information. This information can be collected from all routers over the entire extent of the network and can provide a global view including the origin of the traffic that transits a network. Cross-referencing data from both of these information sources may give a more detailed view that includes the origin of the malicious traffic. The detailed multi-source view provides more context to act upon, and makes countermeasure less sensitive to spoofed traffic information.

This is particularly useful when an attack is volume-based such as in the case of a DDoS attack. In this case, instead of blocking traffic at the end systems, it may be preferable to prevent the malicious data from entering the network at the entry point, or to ask an upstream provider to block the specific traffic. Getting accurate information from the end system is complicated by the fact that this attack traffic can be spoofed to cover its origin. Spoofing causes traffic to have another entry point into the network than the entry point presented to the end system by setting a fake source address. Since the addressing information cannot be relied upon, one has to determine the origin by checking presence of this traffic pattern on all routers on the path.

In our development of CoreFlow we relied specifically on Bro data, on NetFlow information, on Splunk for data aggregation and on Route Explorer for path calculation.

5.3.1 *Bro*

Bro [79] is an open source network analysis framework developed at the International Computer Science Institute in Berkeley, CA and the National Center for supercomputing Applications in Urbana-Champaign, IL. Bro focuses on network security monitoring and offers functionality beyond traditional intrusion detection systems. It includes an event engine and a policy module in which one can write custom policies. Due to clustering capabilities, Bro can scale up to 100Gbps links [14]. Bro has an extensive policy system that can be used to trigger events and to react to them. Events can thus also be correlated within the Bro framework itself as part of a policy. To implement policies Bro uses its own scripting language. This language is limited, but it could be used to implement the CoreFlow functionality as a plugin in the C language. This would require knowledge of two languages, the Bro domain specific language

and C; for this reason it seemed more practical to us to implement CoreFlow as a stand-alone system using Python.

Building a stand-alone system makes CoreFlow more flexible since we are able to use multiple input sources or switch out Bro in favor of a different IDS. Python is a widely used and easy to learn language which became very popular among data scientists. Therefore, by using Python, we try to make it easier for collaborators to extend CoreFlow with new features. Additionally, Python has a large set of libraries and tools available that are specifically useful for analysis and working with large data sets. These libraries can be used to aid the correlation and enrichment process.

5.3.2 *NetFlow and IPFIX*

NetFlow, originally developed by Cisco Systems, but now present on most modern routers is a protocol that allows routers and other network devices to export flow information. According to [75], Cisco traditionally distinguishes a flow based on 7 properties, two of which are not required:

- IP source address
- IP destination address
- source port
- destination port
- L3 protocol type
- Class of service (optional)
- Router or switch ingress port (optional)

These properties are extended in subsequent versions such that NetFlow supports IPv6, vlans, and MPLS labels.

IPFIX (IP Flow Information eXport) described in RFC5153[12] is a protocol developed by IETF that supersedes NetFlow v9. Most software tools and the collectors that work with NetFlow information also accept the IPFIX format. In this chapter we use the term NetFlow to refer to both the NetFlow and IPFIX protocols. The data we import from the routers into CoreFlow uses the *nfdump*[1] format.

5.3.3 *Splunk*

Splunk[15] is a search and analysis system for Big Data, that is often used as a security information and event management (SIEM) system. It can be used to import logs from multiple sources for

[1] nfdump website: https://github.com/phaag/nfdump

analysis. It provides a web interface that can be used to search and to make visualizations of the data for easy analysis. If needed, Splunk can also trigger security alerts. ESnet uses Splunk to aggregate and visualise log data, therefore we set up CoreFlow to consume the already aggregated Bro data in Splunk via a REST interface.

5.3.4 *Packet Design Route Explorer*

Route Explorer[78] is a route analysis system developed by Packet Design. The appliance provides visibility into routing behaviour for IGP and BGP routing protocols and VPNs. By peering with the routers in the network, Route Explorer is able to track real-time changes in logical topology of network; it monitors routing tables and can store them for historical analysis. It can then be used by network administrators to debug problems in a complex network infrastructure. CoreFlow can use Route Explorer to perform path calculation (see Sec.5.5.1).

5.4 COREFLOW ARCHITECTURE

The architecture of CoreFlow is composed of three distinct phases: input, enrichment, and output. This is shown in Figure 5.2.

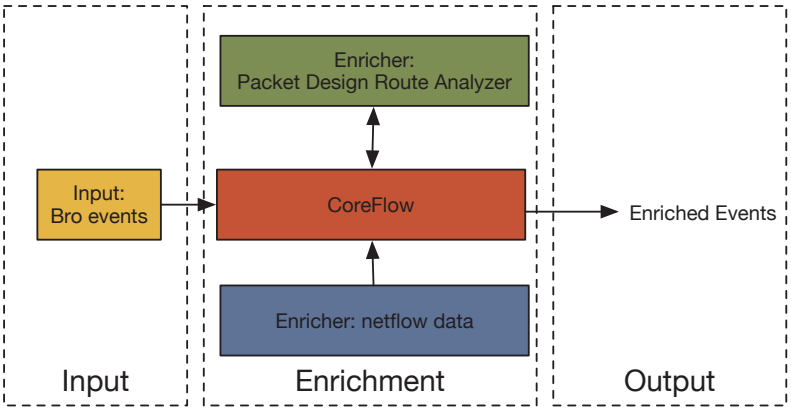


Figure 5.2: CoreFlow correlates input data from Bro to NetFlow and uses the enriched data to query the route analyser. Finally, it outputs the security event with additional data from both enrichers.

The CoreFlow development was driven by a number of design requirements:

- support the Bro data format. The system needs to ingest and process Bro data;
- allow for multiple input sources. We wanted to be able to accept Bro data from different sources, for example reading from file or gathering it in real-time;
- process large amounts of NetFlow data. The system needs to process data from multiple routers;

5.4.1 *Input Phase*

We support multiple ways to import the Bro data into CoreFlow:

FILE operates on Bro log files in either text or gzip format

STDIN operates on output from the standard input in Bro log format

SPLUNK opens a socket to the Splunk server and starts a real time search for incoming events

ELASTICSEARCH reads Bro data that has been imported into Elasticsearch[2] using an included import tool

The *stdin* and *splunk* input methods support streaming of real time data. The *file* and *splunk* methods support reading historical data from within a specified time window. We will elaborate on these two different uses in Sec. 5.4.2.

As main input we use the Bro notice log; this log file contains (security) events that are interesting enough to require further investigation. The fields relevant for correlation are listed in Table 5.2.

The *uid* field contains a unique identifier which is a hash that is based on various properties of the event. The hash can be used to cross-reference the event data between multiple Bro log files. To cross-reference Bro events to NetFlow data we cannot use this *uid* and we are required to match on the flow data contained in the event. Not all Bro events contain the required flow data and the events without this data are passed to the output queue without further enrichment.

We chose to represent the flow information in CoreFlow with a tuple consisting of 5 elements: protocol, source ip, source port,

[2]Elasticsearch is a full-text search engine. https://www.elastic.co/

field	type	description
ts	datetime	timestamp
uid	string	unique id to look up in conn log
id.orig_h	string	ip address source
id.orig_p	string	source port
id.resp_h	string	ip address destination
id.resp_p	string	destination port
proto	string	protocol (TCP,UDP,ICMP)
...	...	...

Table 5.2: Bro notice.log fields necessary for the cross-referencing

destination ip and destination port. These elements correspond to the mandatory NetFlow properties we discuss in Sec.5.3. Each one of these properties corresponds to a specific Bro field. Table 5.3 shows the mapping. Since we are working with data from multiple nodes, event time stamps may not be the same everywhere and are not used in the initial matching process.

CoreFlow	proto	ip1	port1	ip2	port2
Bro	proto	orig_h	orig_p	resp_h	resp_p
NetFlow	pr	sa	sp	da	dp

Table 5.3: The CoreFlow flow tuple and the equivalent fields in Bro and NetFlow data

5.4.2 *Enrichment Phase*

We distinguish two modes of cross-referencing: historical and real time.

Historical cross-referencing specifies a time window in which to match the flows. CoreFlow first processes the Bro data, cross-references it with the NetFlow data and then exits. The sizes of the log files can easily exceed gigabytes; the data workflow is cus-

tomized to minimize memory utilization and random IO and to retain reasonable speeds.

Real time cross-referencing works by streaming the latest events from the Bro notice log. Since we are using *nfdump* files for NetFlow processing and do not have a source that was able to stream real time NetFlow information, there is a time delay introduced in processing the events. CoreFlow periodically sends out NetFlow searches and queues events until the previous search is completed, this approach prevents slowdown caused by many searches blocking on disk I/O.

After the matching process, the Bro event is enriched with one or more NetFlow records, one for every router it was seen on. When combined with sufficient topology information, one can now estimate the exact path of the event flow and of the ingress and egress router and ports (see Sec. 5.5.1).

5.4.3 *Output Phase*

When the NetFlow and path information is merged with the Bro events, summary output is written to *stdout*. Additionally, CoreFlow provides a simple output module which exports enriched output as JSON to a log file. There is also experimental output support to Elasticsearch. Since the framework is extendable, we also consider other outputs. For example, we consider a Bro output that can be used to feed the enriched output back into Bro to create new alerts. This idea is discussed in Sec. 5.6.

5.5 IMPLEMENTATION

The first prototype of CoreFlow is implemented in Python 3.5 using the Python requests and Elasticsearch libraries. CoreFlow has a main loop that routes messages from an input to an output via the NetFlow enricher.

Figure 5.3 shows the execution flow of CoreFlow. There are three threads: a main loop, an input thread, and a search thread.

The input modules run in a separate thread that is being watched and when necessary restarted by CoreFlow. CoreFlow receives the data from the import thread via an event queue which contains the *bro_alert* and the *event_id*.

CoreFlow reads the events queue and when it finds a new event, it extracts the flow tuple. The flow tuple, together with the event id, is inserted in a queue for the NetFlow enricher. When the NetFlow enricher is idle, it picks up all items in the queue at once; it creates a

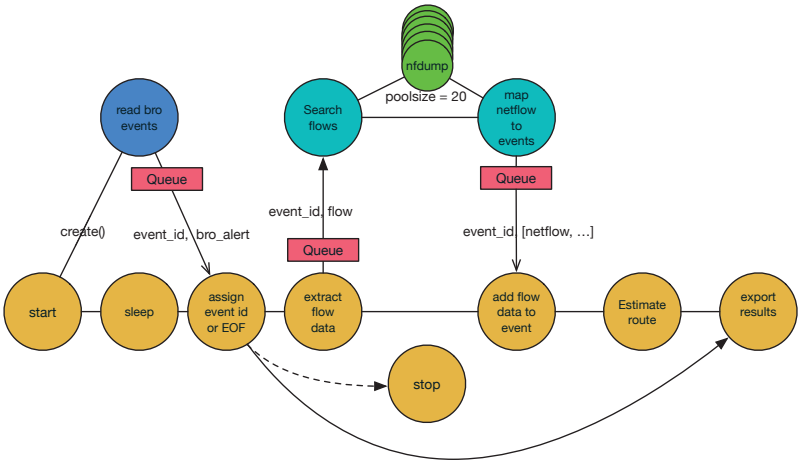


Figure 5.3: Execution flow of CoreFlow, with its three threads: main loop (orange), input thread (blue), search thread (cyan/green).

filter for all the flows and their reverse that can be passed to *nfdump*. A reverse flow is simply the flow detected by Bro with source and destination IP/port swapped. We use both the (forward) flow and its reverse because we want to have visibility in both directions of the traffic. Creating such a bulk request is noticeably faster than requesting each event one by one because now we have to search through the flow data only once. Depending on the amount of routers in the network, the NetFlow enricher will spawn one search thread per router that runs *nfdump* with the previously compiled filter. The results of the bulk request comes back out of order, thus we need to re-order and map the retrieved flows back to the original Bro data.

Now the NetFlow data is mapped back to the event identifiers, it gets inserted into another another queue for further processing in CoreFlow. CoreFlow reads the NetFlow data from the queue and it combines the existing event data with the NetFlow data. Finally, CoreFlow passes the enriched data on to the output module that forwards it to the storage backend.

The enriched event-data can contain multiple occurrences of the flow, reported by multiple routers. Combined with topology information, CoreFlow tries to reconstruct the path of the flow with a route estimation procedure (see Sec.5.5.1). For more detailed route estimation, CoreFlow can interface with products such as Route Explorer by Packet Design.

Finally, after the routes have been identified, CoreFlow exports the results.

5.5.1 *Route Estimation*

ESnet uses OSCARS [37] for provisioning links across its network and OSCARS therefore maintains a database with topology information. To create the required topology information for CoreFlow, we extract the topology information from the OSCARS topology publisher. The extracted information does not contain policy information or any of the routing metrics that are used to set a preference for certain paths. Therefore, we decided upon finding the shortest path with the constraint of traversing all the routers for a single flow as an approximation. We designed Algorithm 1 with the following requirements in mind:

- the input list may have missing routers; a flow may traverse a router but may not be recorded due to the sample rate.
- the path may traverse a router multiple times; flows may be observed on a router twice using different vlan/mpls labels.

The algorithm works as follows:

As a starting point we take the first router in list D, *start* (line 5); then we use the *topology* to build a tree from *start* limited to a *depth* and return the paths as an array P (line 6). To include all possibilities the depth should be set to the maximum spanning tree distance of the network graph.

We reverse all the paths (lines 7-9) and then we concatenate the result R with the original paths in P (lines 10-14). This gives us list A of all paths that traverse the *start* node. We then filter A to only include paths that contain all routers in D and store this as F (lines 15-18). We select the shortest paths in F and return this list as value O (lines 20-23).

The output of this algorithm can be illustrated with a simple example. Figure 5.4 shows a *topology* with nodes $r1 - r12$; a flow entered the network at $r2$ and exited at $r10$. The routers which observed the flow are $D = [r1, r12, r3]$. Our algorithm is able to interpolate that $r4$ and $r9$ are part of the path and it returns $[r1, r4, r3, r9, r12]$ as the estimated route together with its reverse, $[r12, r9, r3, r4, r1]$. Note that $r2$ and $r10$, the ingress and egress nodes, are not part of the reconstructed path as they had not observed the flow themselves directly. A current limitation of the algorithm is that is not capable of determining which router was the actual ingress and egress

Algorithm 1: route estimation algorithm

Input: *topology* ← topology graph of the network
Input: *depth* ← max search depth
Input: D ← detected routers in the path
Output: list of estimated paths (O)
start ← $D[0]$;
P ← all paths with length $\leq$ to *depth* starting from *start* in the *topology*;
foreach $p \in P$ **do**
    $R \leftarrow$ add reverse(p);
end
foreach $p \in P$ **do**
    foreach $r \in R$ **do**
        $A \leftarrow$ add $r + p[1:]$;
    end
end
foreach $p \in A$ **do**
    if $D \subseteq p$ **then**
        $F \leftarrow$ add p;
    end
end
foreach $p \in F$ **do**
    $O \leftarrow \min(lenght(p))$;
end

router; this is because the topology information we use does not distinguish edge routers.

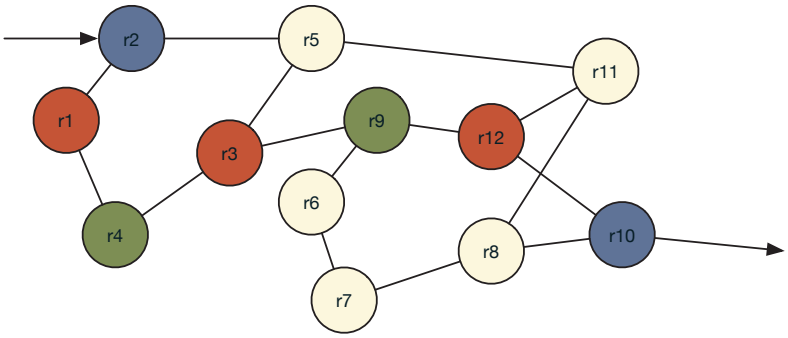


Figure 5.4: Route estimation: given a list with routers [r1 r12 r3] the algorithm is able to interpolate that r4 and r9 must also be included in the path resulting in [r1, r4, r3, r9, r12]. However, with the current information, the algorithm cannot deduct the edge routers r2 and r10 should be included in the path.

5.6 EVALUATION AND DISCUSSION

We tested the prototype on the ESnet infrastructure by enriching incoming events from three different Bro nodes with NetFlow data collected by over 50 routers. Figure 5.5 shows the latest set-up we used for CoreFlow at ESnet. There were two specific limitations in ESnet that we had to deal with.

Firstly, we had 3 Bro detectors sending their logs to Splunk. CoreFlow was reading the logs from Splunk and performing searches on NetFlow data of all routers. The NetFlow data was exposed to CoreFlow via an NFS share. Every 5 minutes, a new NetFlow log of a router gets saved and gets copied to the NFS server. Under normal circumstances, copying the NetFlow data takes less than 3 minutes using this setup. This meant that we had to delay the retrieval of incoming events from Splunk by $5 + 3 = 8$ minutes.

Secondly, the flow is detected on multiple routers, CoreFlow performed route estimation as described in Sec. 5.5.1 and prepared queries for the Route Explorer to further refine the found route. Due to access restrictions we could not query the Route Explorer

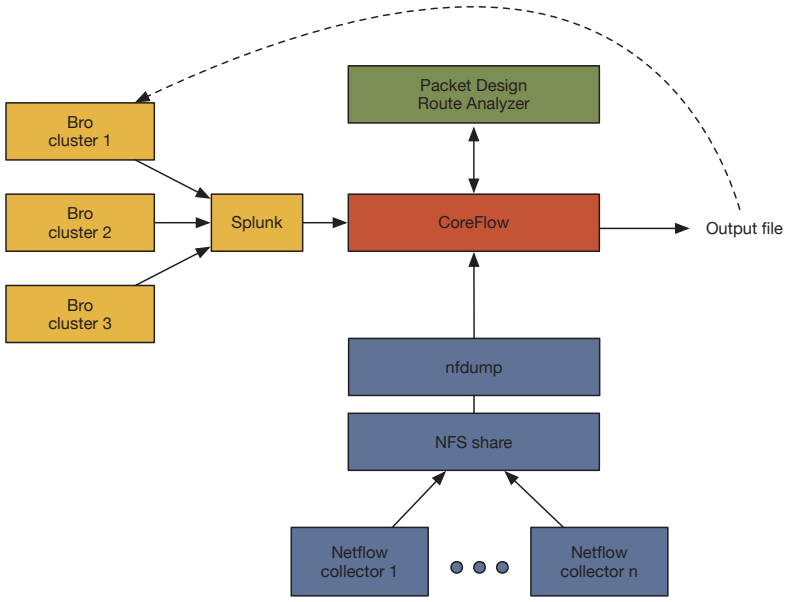


Figure 5.5: CoreFlow set-up at ESnet

directly, thus we verified this functionality by sending the query, manually, from another host.

5.6.1 *Route Estimation*

The route estimation can be optimised in multiple ways. The data structures contain redundant information, and for large networks this data structure may get too big. The algorithm does not deal with metrics and routing policies and any traffic engineering that can manipulate the flow of traffic because this information was not available at the time. Improvements to the route estimation can be made by calculating paths based on live routing tables of the network. For historical paths we can rely on products such as Packet Design's Route Explorer that records changes in the routing table over time. By recording this information, Route Explorer can provide paths from an ingress router to a destination prefix at any point in time. However, this requires us to determine the ingress router of the specific flow and when NetFlow traffic is sampled we may not be able to see the flow on the ingress router. If we find the flow on one or more routers, in some situations we can use the

route estimation explained in section 5.5.1, to extrapolate a potential ingress router that we can use for the full path calculation.

Adding reconstructed paths and NetFlow information to security events allows for more targeted monitoring or mitigation techniques e.g. blocking at the source or redirecting the traffic somewhere along the path for further analysis. Additionally, one can feed the enriched data back into the IDS to enhance filtering on relevant alerts e.g. by lowering the threshold for data going to CoreFlow and create more specific event filters on CoreFlows output.

5.6.2 *Sample Rate*

Another point of attention is the sampling rate of NetFlow. In ESnet, for example, the sampling rate of the data was set to 1:1000 on each router. The unfortunate side-effect of a low sampling rate is that the probability to find flows related to the IDS alerts is also very low, since the sampling rate needs to be multiplied by rate of malicious flows to all traffic on each router. This can be improved by increasing the sampling rate on all the routers. In ESnet, increasing the sample rate was not possible since ESnet is a production network and higher sampling rates can result in degraded network performance, because a high sampling rate require more processing on the production routers. Another way to improve the chance of finding flows with less impact on the network is by using higher sampling rates at the edges. This approach may be feasible in carrier networks, since the bulk traffic streams are located in the core. Additionally, this approach also increases the chance of finding the flow on the ingress router, which benefits path estimation and can help to apply counter measures at the point of entry. Methods described in [2] can also help improve the sampling algorithms in to detect smaller flows.

One might argue whether or not it is necessary to increase the sampling rate to detect small flows on the network in the context of network security: Volume based attacks such as DDoS attacks will, for example, clearly be visible in sampled data. Yet given the right circumstances, an attacker can do a lot of damage using only a few packets. Moreover, there are instances on attackers using volume-based attacks to distract the victim from the real attack [68]. Therefore, it is important to provide as much context as possible, to every event that the intrusion detection system marks as malicious.

5.6.3 *Other Use-cases*

CoreFlow can also be used in multi-domain defence strategies. When the ingress point of the spoofed malicious traffic can be identified, it is possible to contact the neighbouring domain to take action. If the neighbour also has such a system, it can subsequently contact its neighbours, eventually tracing the traffic back to the source. Taking action closer to the source of the problem can unburden networks of volume-based attacks and keep the resources available for the legitimate users.

In Sec. 3.5 we concluded that responses can become complex and even counter intuitive when networks increase in size and when information is limited. The SARNET-agent in Sec. 2.2 can greatly benefit from CoreFlow since it provides richer information and the context to improve classification and enhance the decision making.

5.7 RELATED WORK

Much work is done on applying statistical methods and machine learning approaches to NetFlow data in order to detect anomalous behaviour on computer networks. These anomalies can be caused by network changes, outages, content changes or security related events. Sperotto et al. published a comprehensive overview of flow-based intrusion detection in [97].

CoreFlow's goal is not to identify security threats. CoreFlow does not perform any intrusion detection. It assumes that there are facilities in place that generate these security events. CoreFlow's uniqueness is that it focuses on the correlation and enrichment of already identified events by using multiple data sources such as NetFlow, and topology databases, to create a more comprehensive view of what occurred in order to enhance decision making.

Xu et al. describe a system that can group low-level events from several inputs based on similarities or relations[113]. When multiple low-level events trigger at the same time, they can be grouped into a more meaningful high level event. This high level alert can be created to trigger a defence. Our approach is different, since we cross-reference the triggered events to other data sources that may not have generated alerts themselves in order to expose more contextual information for further analysis. If we would accept multiple input sources, then grouping triggered events becomes relevant for CoreFlow, yet this is considered future work.

5.8 CONCLUSION AND FUTURE WORK

Enriching IDS data with NetFlow information gives a more detailed record of an attack. CoreFlow provides a framework that can cross-reference these data sources based on the flow tuples. The successfully enriched data can be used for more advanced attack detection and reaction.

We determined that the success of the NetFlow correlation largely depends on the sampling rate of the NetFlow data. We showed how to use the enriched information to do route estimation. Route estimation can be used for carrier networks to determine where the traffic entered their network and can be the starting point for placing countermeasures close to the origin of the attack. The route estimation method also decreases sensitivity to spoofed traffic because the NetFlow data confirms that the traffic got forwarded by the devices in the estimated route.

CoreFlow needs to be evaluated using different sample rates (1:1) and other sampling algorithms to see which settings are most beneficial, while not affecting the performance of a production network. To address the sampling issue, we are looking at new ways of flow tracking: Hill et al. [40] research whether bloom filters implemented in P4, a language for programming data plane switches, can provide a solution to this problem.

CoreFlow can be extended to allow multiple in- and output plugins for other data sources such as PerfSonar[3], and syslog. CoreFlow also supports extensions that contain new analysis methods which can help to interpret the information and provide more insights into the context of an event.

The enriched event can maybe lead to improved and more advanced alerts. By feeding the enriched event data back into the IDS system, it may be able to use the extra information to reduce false positives. Also, it may be beneficial to lower the thresholds for IDS events that get sent towards CoreFlow to discover malicious activities that previously went undetected.

[3] PerfSonar website: https://www.perfsonar.net/

6

APPROACHES FOR COLLABORATIVE MULTI-DOMAIN DEFENCE AGAINST ATTACKS ON COMPUTER INFRASTRUCTURES

Some attacks accumulate, due to their distributed nature, so much power that they cannot be stopped at the border of a network. In Chapter 4, we concluded that we need multi-domain collaboration to ask the upstream providers of the attacked network for assistance in defence. This chapter extends the VNET environment to support multiple domains. Each domain is autonomously controlled by a SARNET-agent, which is extended to communicate and orchestrate defences in an alliance environment. We developed three different approaches to defend against an attack. Each approach uses a different algorithm to execute the same defence tasks at the selected members in the alliance. By evaluating the efficiencies of the approaches under different circumstances, and by analysing what caused those differences, we answer RQ3 by pointing out the factors that play a role in multi-domain defences.

This chapter is based on:

- **R. Koning**, G. Polevoy, L. Meijer, C. de Laat, and P. Grosso *"Approaches for Collaborative Security Defences in Multi Network Environments"* [51], in *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, ©.

6.1 INTRODUCTION

Resolving distributed attacks on computer infrastructures often benefit from the support, resources, and actions of parties other than the victim.

To facilitate cooperation, Deljoo et al. [28] introduce the concept of security alliances, a framework in which groups of organisations can help each other to defend against attacks on one of the members in the alliance.

Security alliances [30] provide certain benefits and services such as:

- establishing and maintaining trust among the members,

- facilitating governance and common policies, standards for its members,
- creating a platform for sharing threat intelligence and incident information among members, and
- supporting coordinated defense mechanisms.

When asking for help from other members in an alliance, even when asking for the same defensive actions, defence performance may differ based on the implementation approach.

The question is: *How do different approaches for the same defence affect the efficiency of the defence in alliances consisting of multiple network domains?*

In this chapter we introduce three approaches that use collaborators in an alliance for the same defence against an attack:

- *Counteract Everywhere* mitigates the attack at every collaborator;
- *Minimise Countermeasures* reduces the amount of placed countermeasures by only mitigating close to the attacker;
- *Minimise Propagation* mitigates as closely to the victim as possible, and reduces countermeasure propagation.

We extended the VNET infrastructure discussed in Chapters 3 and 4 and added support for multiple domains. Each domain is monitored and controlled using its own agent. The agents communicate with other domain agents to defend collaboratively, using one of the three approaches mentioned above. The agents act autonomously using the resources available in the domain, without detailed information from other domains.

In Chapter 4 we introduced a method to evaluate the efficiency of defences in single domain environments. In Sec. 6.4, we provide an extended formula for efficiency to include support for multiple factors that influence the efficiency. We use this formula to compute the efficiency of our multi-domain approaches.

We implement these approaches in the SARNET environment and evaluate the efficiency of the approaches during attacks with varying conditions. We will argue which approach is the most efficient in most situations and that care should be taken in the development of these countermeasures.

The chapter is organised as follows: In Sec. 6.2 we explain the SARNET framework we use for this work. Then, we describe the

three different defence approaches in Sec. 6.3 and explain the extended efficiency formula in Sec. 6.4. In Sec. 6.5 we explain how the approaches are implemented in VNET, that we use to run our experiments. We discuss the attack scenario in Sec. 6.6 and in Sec. 6.7 we show the scenarios and conditions we used for the experiments of which we will show the results in Sec. 6.8. Finally, we discuss our findings in Sec. 6.9 and conclude and elaborate on future work in Sec. 6.11.

6.2 MULTI-DOMAIN SARNET

In Chapters 2 and 3 we introduced SARNET as a framework for detection and mitigation of attacks on computer infrastructures, i.e. computer systems that are interconnected using networks to provide a service. The framework collects *metrics* from both the network, the systems connected to the network, applications running on top of systems, and performance metrics from higher level systems (such as the amount of products that are sold during a time interval). An *observable* adds a condition to the metric that can be monitored to see if the metric deviates from either, a threshold, a ratio or a level based on historical values. When the deviation is too large, depending on the condition, the observable changes from a *healthy* to an *unhealthy* state. One or more observables map to a *classification*; based on the classification further analysis may happen, including database look-ups, to collect the necessary information in order to decide which defences to pick. When multiple defences are available for the situation, the SARNET picks the *defence* with the highest efficiency ranking and executes it. A defence consists of multiple *tasks*. After executing defensive actions, SARNET evaluates the metrics again and recalculates the efficiency rankings. If the system performance is still affected by the attack, the cycle repeats and another defence is tried until all possibilities are exhausted. When the possibilities are exhausted, the system will remain in a degraded state until the attack ends or is resolved by human intervention.

To allow SARNET to operate in a multi-domain situation, each domain needs a separate agent. Agents are responsible for coordinating activities between collaborators in the alliance. These activities include coordinated responses to threats as well as information sharing, such as threat intelligence or analytics data, with agents of other domains.

6.2.1 *Defence Orchestration*

Even when collaborating within an alliance, a member might choose not to share sensitive data with others, because of company policies, or because laws and other norms [7] prevent them from doing so. For these reasons, we used a decentralised approach for defence orchestration. In our architecture there is no central authority with a full overview of the alliance, and the domains themselves are responsible for building their own overview of the alliance and the networks surrounding it. We default to limited information sharing between the parties; the amount of information sharing can of course be increased when the situation requires it. Another advantage of this decentralised approach is the increase in robustness against attacks since there is no single point that can be attacked to cripple defence orchestration.

6.2.2 *Responsibility*

It is the victim's responsibility to detect and classify the attack and decide on the action that needs to be taken. Only the victim has the full view of its infrastructure and knows when it is truly under attack. Placing the responsibility on the victim also prevents disputes if a collaborator accidentally disrupts benign traffic since the defence is activated at the request of the victim. Of course, the potential victim is allowed to delegate some responsibilities to another party when necessary; handling the potential issues that arise from delegating responsibility is beyond the scope of this work.

6.2.3 *Agent Communication*

Each member of the alliance runs a multi-domain agent. The multi-domain agents establish secured connections with all the other agents in the alliance to send messages directly between domains. Communicating directly removes the complexities of maintaining message integrity. We distinguish four different kinds of messages:

- *Control messages* for setting up and maintaining communication to other domains.
- *Informational messages* for requesting information and responding to the requests.

- *Action messages* for implementing a certain countermeasure to reduce the attack impact.
- *Subscriptions* for longer lasting information or actions such as intelligence feeds or automatic protection.

Note that informational and action messages can result in simple queries or actions limited to the destined domain, but also to more complex queries or actions where the destined domain can request help from others. The countermeasures in this chapter use only simple direct actions: 1) requesting from which neighbour traffic of a certain pattern originates and 2) dropping the traffic that matches that pattern.

6.3 INTER-DOMAIN DEFENCE STRATEGIES

We defined three defence strategies for attacks that rely on cooperation between alliance members:

- Algorithm 2 - Counteract Everywhere
- Algorithm 3 - Minimize Countermeasures
- Algorithm 4 - Minimize Propagation

Algorithm 2 shows the most aggressive approach. The approach immediately implements a countermeasure in the victim's domain as well in all other collaborating domains if there is a pattern match. This results in a small impact because the countermeasure is applied as soon as possible, but since the countermeasure is applied on all alliance members this operation is costly (see Eq. (6.1)):

$$
\begin{aligned}
cost &= \sum_{n=1}^{N} C_n + P_n \times t \\
\text{where } N &= \text{participating nodes} \\
n &\in N \\
C_n &= \text{ fixed cost of node } n \\
P_n &= \text{ cost of node } n \text{ per time period} \\
t &= \text{amount of time periods elapsed}
\end{aligned}
\tag{6.1}
$$

Algorithm 3 reduces cost by only implementing countermeasures at members that see the attack pattern coming in from non-members. Since the nodes at the edge of the alliance E are a subset of all nodes

Algorithm 2: Counteract everywhere: asks for countermeasure from every member that sees atack traffic

Input: *pattern*: attack pattern
alliance: alliance members
N : victim's neighbours
for *node* $\in N$ **do**
request *node* for its *neighbours* that produce *pattern*;
implement countermeasure at *node*;
for *neighbour* $\in$ *neighbours* **do**
if *neighbour* $\in$ *alliance* $\wedge$ *neighbour* $\notin N$ **then**
add *neighbour* to N;
end
end
end

N implementation costs (see Eq. (6.2)) are equal to or less than in Algorithm 2:

$$
\begin{aligned}
cost &= \sum_{e=1}^{E} C_e + P_e \times t \\
\text{where } N &= \text{participating nodes} \\
E &\subseteq N \\
e &\in E \\
C_e &= \text{ fixed cost of edge node } n \\
P_e &= \text{ cost of edge node } n \text{ per time period} \\
t &= \text{amount of time periods elapsed}
\end{aligned}
\tag{6.2}
$$

The disadvantage of using this approach, though, is that the time to implement a defence increases; the victim domain first has to trace the attack origin back to the edges of the alliance, before it can ask the edge domain to implement any countermeasure.

Algorithm 4 is another optimisation of Algorithm 2. This approach reduces cost by reducing propagation, relying on recovery detection. The approach directly applies the countermeasure at the neighbours, but instead of going to the neighbours of the neighbours directly, it first waits for a time period defined by *wait time*. Only if the attack is not resolved, the approach request the nodes' neighbours for assistance. Because the approach tries to minimise the amount of assistance, the implementation costs will be lower

Algorithm 3: Minimise countermeasures: places the countermeasures close to the attack traffic at the border of the alliance.

Input: *pattern*: attack pattern
alliance: alliance members
N : victim's neighbours
for $node \in N$ **do**
  request *node* for its *neighbours* that produce *pattern*;
  if $\exists neighbour \notin alliance$ **then**
    implement countermeasure at *node*;
  end
  for $neighbour \in neighbours$ **do**
    if $neighbour \in alliance \wedge neighbour \notin N$ **then**
      add *neighbour* to N;
    end
  end
end

(worst case they are equal to Eq. (6.1)). However, by waiting until we reach *wait time* while under attack the impact continues to increase.

The defence time for Algorithm 4 can improve when the *wait time* is set to a smaller value. However when the *wait time* is set too small, i.e. when it is smaller than the amount of time that the system requires to detect recovery, the approach will continue asking other nodes. In other words, setting *wait time* too small causes Algorithm 4 to behave similarly to Algorithm 2, but with a time penalty caused by *wait time*. Ideally, *wait time* is tuned to the time it takes the victim to reliably detect recovery. This implies that the faster the victim can reliably detect recovery, the more efficient Algorithm 4 becomes.

6.4 EFFICIENCY

For each defence approach, we want to assess its efficiency as function of relevant metrics in each scenario: there will be parameters that increase the efficiency, and parameters that decrease its value.

For example, during or after an attack, we observe the deviation of each relevant metric from its baseline prior the attack. We define *impact* as the accumulated deviation from the start of the defence until the time of evaluation t. The higher the impact is, the lower the efficiency should be. On the other hand, efficiency is also con-

Algorithm 4: Minimise propagation: minimises propagation of the countermeasure by filtering close to the victim.

Input: *pattern*: attack pattern
alliance: alliance members
N : victim's neighbours
resolved: true when the attack is resolved otherwise false
waittime : time to wait for the system to evaluate its attack state
for *node* $\in N$ **do**
  request *node* for its *neighbours* that produce *pattern*;
  implement countermeasure at *node*;
  wait for *time* seconds;
  if *attack not resolved* **then**
    for *neighbour* $\in$ *neighbours* **do**
      if *neighbour* $\notin N$ **then**
        add *neighbour* to N;
      end
    end
  end
end

strained by the available budget to implement a countermeasure. The less amount of budget spent, the higher the efficiency becomes.

We evaluate the performance of the defence approaches by generalising Eq. (4.1) from Chapter 4. In the original formula, the efficiency decreased in both allowed parameters. Now, we expand those formulas to allow for any finite number of parameters. Given a strictly increasing function f, such that $f(0) = 0$, we assume that efficiency decreases in $f(x_j)$ and increases in $f(y_i)$.

In our efficiency formula, we normalise the contribution of these parameters as follows:

- For factors that decrease the efficiency, x, we normalise to values from 0 to 1 as follows: $\frac{f(X)-f(x)}{f(X)}$.
- For factors that increase the efficiency, y, we normalise to values from 0 to 1 as follows: $\frac{f(y)}{f(Y)}$.

The efficiency formula utilises the following symbols:

- β sets the division point between no recovery $[0 - \beta]$ and recovery $[\beta - 1]$.

- The efficiency increasing factors are denoted as $y_1, \ldots, y_m$; m is the amount of decreasing factors.
- $x_{m+1}, \ldots, x_{m+l}$ denote the factors that decrease efficiency; l is the amount of increasing factors.
- The importance α generalises to $\alpha_1, \alpha_2, \ldots, \alpha_{m+l-1}$. The parameters α_i fulfil that $\sum_{i=1}^{l+m-1} \alpha_i$ is between 0 and $1-\beta$. The last factor, α_{m+l}, then implicitly gets the importance of $1 - \beta - \sum \alpha_1 \ldots \alpha_{m+l-1}$.

To use the efficiency formula, we assume that there is at least one factor (x_{m+l}) present that decreases the efficiency, without loss of generality.

The efficiency is defined as follows, for the case where we recover from the attack and the one where we do not:

$$E(\text{recovered or not}, y_1, \ldots, y_m, x_{m+1}, \ldots, x_{m+l}) \triangleq$$

$$\begin{cases} \beta + \sum_{i=1}^{m} \alpha_i \frac{f(y_i)}{f(Y_i)} + \sum_{j=m+1}^{m+l-1} \alpha_j \frac{f(X_j)-f(x_j)}{f(X_j)} \\ +(1-\beta-\sum_{k=1}^{m+l-1} \alpha_k) \frac{f(X_{m+l})-f(x_{m+l})}{f(X_{m+l})} & \text{Recovered,} \\ \sum_{i=1}^{m} \alpha_i (\frac{\beta}{1-\beta}) \frac{f(y_i)}{f(Y_i)} \\ + \sum_{j=m+1}^{m+l-1} \alpha_j (\frac{\beta}{1-\beta}) \frac{f(X_j)-f(x_j)}{f(X_j)} \\ +(1-\beta-\sum_{k=1}^{m+l-1} \alpha_k)(\frac{\beta}{1-\beta}) \frac{f(X_{m+l})-f(x_{m+l})}{f(X_{m+l})} & \text{otherwise.} \end{cases} \tag{6.3}$$

The full characterisation of efficiency is provided in [81].

Equation (6.4) in Sec. 6.7 shows a practical example of how we use Sec. 6.4.

6.5 IMPLEMENTATION

To evaluate the three proposed approaches and their effectiveness we used the SARNET framework. In the following sections, we first introduce VNET (Sec. 6.5.1), the elements in our topologies (Sec. 6.5.2), the inter-domain signalling protocol used by the collaborators in the alliance (Sec. 6.5.3), and the SARNET/VNET implementation of the three algorithms (Sec. 6.5.4).

6.5.1 *VNET*

We use the VNET emulation environment to instantiate the topologies in Sec. 6.7. VNET instantiates the topology as a network slice [115] on the ExoGENI cloud platform [3]. Each virtual machine

on the ExoGENI platform runs a single domain in our multi-domain setup. We use two VM types: XOSmall (1 core, 1G RAM) for customer and transit domains and XOLarge (2 cores, 6G RAM) for the NFV and service domains. ExoGENI ensures that the requested link capacity between virtual machines is guaranteed by preventing overprovisioning and limits the bandwidth to the requested 100 megabit per link. The various components inside the domain are separated in Linux containers [19], using Docker [1]. All the components within a domain communicate to each other using MQTT [2]. Routing between the domains is done with the Quagga software router [43] using the BGPv4 routing protocol [87]. Each domain runs an agent that talks to other domain agents and communicate with our controller that configures and executes the attack scenarios. Defences are started autonomously, when an attack is detected by the local SARNET agent that runs in each domain.

6.5.2 *Topology Building Blocks*

Using VNET we can construct a virtual infrastructure by supplying a topology with the following components:

- a *service domain* contains a webservice that resembles a marketplace where clients make purchases;
- a *transit domain* forwards traffic, it provides basic blocking, redirection and rate limiting functions;
- a *client domain* interacts with the service domain by making transactions with the service domain;
- a *NFV domain*; runs network functions, using Network Function Virtualisation, that are used for further analysis or countermeasures.

Traffic flows back and forth from the *client domains* via the *transit domains* towards the *service domain*. Normally, the traffic consists of transactions (simulated purchases) with the *service domain*. When we start an attack we instruct one or more *client domains* to attack the victim which is the *service domain*. *NFV domains* can be used to run defensive network functions through which the traffic can be routed. Although *NFV domains* are present in the topology they have not been used in these experiments since the available functions are not relevant for the attack scenario (Sec. 6.6 we use in this chapter).

[1] https://docker.io
[2] http://mqtt.org

6.5.3 *Inter-domain Signalling Protocol*

Each domain runs a multi-domain agent that handles communication between domains (see Sec. 6.2.3). The multi-domain agent is responsible for keeping track of multi-domain communication and coordinating with the local SARNET agent; as well as other multi-domain agents to orchestrate multi-domain defences.

The main responsibilities of the multi-domain agents are requesting defences and communicating queried metadata that can be useful for a defence, e.g. "Do you see traffic coming from this IP address?".

In the current implementation, agents cannot forward messages that they receive from other agents, therefore all agents need to able to reach each other directly. The agents communicate over Transport Layer Security (TLS), which takes care of authentication and encryption. The messages that they exchange over the secured connection are formatted using JSON [3]. Another limitation is that an agent can currently only be part of a single alliance.

The multi-domain agent exchanges messages of the types described in Sec. 6.2.3 and are specifically:
Control messages:

- **Identify**: exchanges information about the domain's identity and which neighbours it has (for topology building)

Informational messages:

- **Ask**: ask another agent if it sees traffic matching a pattern (source address, destination address, protocol type, minimum traffic rate);
- **Match**: positive response to *ask* message with a list containing the neighbours from which the traffic is seen, including ingress or egress indication;
- **NFV Alive**: notify the requesting domain that the NFV has received (attacker) traffic.

Action messages:

- **Deploy**: ask domain to deploy changed link-rate, rate limiter, firewall rule, or to deploy an NFV container of a specific type;
- **Redirect**: ask domain to redirect traffic over a different link;
- **Cancel**: remove a deployed countermeasure.

[3] JavaScript Object Notation: `https://json.org`

6.5.4 *Algorithm Implementations*

The SARNET multi-domain agent allows domains to share intelligence and to collaborate with each other on both resolving and gathering information about attacks.

Using flow information, the origin of an attack can be traced within each domain using the method described in Chapter 5. To trace the origin in a multi-domain environment, the victim asks alliance members whether or not they see the attack traffic. The aiding domains identify which of their neighbours is sending the attack traffic, and they return this information to the victim. When those neighbours are in the alliance, we can repeat this process until we traced back the attack throughout the alliance to its border. Since the domains return where the traffic has been seen, and not where the traffic pretends to be coming from, the defences work correctly even when traffic is spoofed.

Since we use a stateless approach in our network, the victim does not rely on other members to send notifications when the defence is implemented or when the attack has changed. Therefore, the victim periodically sends information requests to see whether or not the attack is still ongoing. If this is the case, the victim also continues to send action tasks to the defending member, for as long as the attack persists. The continuing transmission of action tasks also updates the defences when the attack characteristics change.

6.6 ATTACK SCENARIO

A Distributed Denial of Service (DDoS) attack is a good example of an attack that requires a collaborative response. Zagar et al. stipulate in their conclusion that collaboration, cooperation and distributed defences are key in defending against DDoS attacks [116]. Most of the time, the DDoS attack causes congestion on the link from an Internet Service Provider (ISP) to the victim. For mitigation to have effect, the victim has to ask their ISP to take action. Depending on the scale of the attack, the bottleneck can also exist beyond the ISP's control, in which case other parties need to be included in resolving the problem.

We simulate this attack within the VNET topology by sending many small UDP packets at a chosen rate originating from the client domains. The amount of networks from where the DDoS can originate is constrained by the amount of client domains we requested in our topology, which is limited by the resources available on the

ExoGENI rack. Therefore, the attack does not contain the amount of nodes and the bandwidth that are typical for a large DDoS attack. Still, the attack has multiple origins that need to be blocked in order for defence to have effect so the methods in this chapter apply when they are optimised for scale.

We classify an attack as a DDoS when the congestion observable becomes unhealthy. The congestion observable is based on two other observables: *rxBandwith* and *sales*. *rxBandwidth* is the used bandwidth measured at the entry point of the service domain, while *sales* is the number of successful transactions to the service provided by the service domain. *rxBandwidth* becomes unhealthy if the amount of incoming traffic on the link to another domain exceeds 92 percent of the total link capacity. *Sales* triggers when the amount of transactions to the web service diverges negatively from the expected amount of sales.

When the attack is classified as DDoS, there are two solutions: A Local solution where the victim starts filtering the traffic pattern within its own domain and a Remote solution using the three approaches discussed in Sec. 6.3.

6.7 EVALUATION

We evaluate the strategies mentioned in Sec. 6.3 using the generic efficiency formula (see Sec. 6.4)). The parameter β defines the cutoff point between recovered and not recovered. Since we are evaluating how well defences recover the attack, we decided to assign a larger range to the *recovered* situation than to the *not recovered* situation. Therefore, we chose a β of 0.2 to allow the efficiency in the *recovered* situation to be in the range from 0.2–1.0.

As discussed in Sec. 6.6, we could use the metrics *sales* and *rxBandwidth* to calculate their impacts. The impact on *sales* should decrease efficiency, and the impact on *rxBandwidth* should increase efficiency since less *rxBandwidth* causes less congestion. However, we observed that *rxBandwith* is actually an unreliable metric for evaluation: in fact, an increase in *rxBandwidth* can indicate both a negative (DDoS attack) as well as a positive (sudden increase in customers/sales) effect. For this reason, we set the weights in the efficiency to $W_{rxBandwidth} = 0$ and $W_{sales} = 1.0$ to make sure *rxBandwidth* does not influence the efficiency.

These weights do not include *cost*. We consider the combined impacts, *rxbandwidth* and *sales*, and the *cost* to be equally important.

Therefore, we multiply each weight, $w_{rxbandwidth}$ and w_{sales}, by 0.5 such that *cost* can use the remaining 0.5.

Finally, we multiply the weights with $1-\beta$, to ensure that the sum of each weight including costs equals $1-\beta$ and that the weights can be used as α in the efficiency (Formula 6.3).

In a *recovered* scenario, Formula 6.3 reduces to the following for the attack scenario we chose:

$$
\begin{aligned}
E_{rec} &= \beta + \alpha_S \frac{S-s}{S} + \alpha_T \frac{t}{T} + (1-\beta-\alpha_B)\frac{B-b}{B} \\
\text{where } S &= \text{max sales impact (no sales)} \\
s &= \text{actual sales impact } sales \\
\alpha_S &= \text{importance of sales} \\
&= W_{sales} \times 0.5 \times (1-\beta) \\
T &= \text{max incoming traffic, link capacity} \\
t &= \text{actual incoming traffic, } rxBandwith \\
\alpha_T &= \text{importance of traffic} \\
&= W_{rxBandwidth} \times 0.5 \times (1-\beta) \\
B &= \text{budget} \\
b &= \text{budget spent, or } cost \\
\alpha_B &= \text{importance of budget} \\
&= 1-\beta-(\alpha_S+\alpha_T) \\
e_{rec} &\rightarrow [\beta, 1]
\end{aligned}
\tag{6.4}
$$

This is the equation we will use in the rest of our evaluation.

6.7.1 *Evaluation Conditions*

We vary the following conditions in our evaluation: topology size, alliance size, budget available at each domain, and the attack load.

6.7.1.1 *Topology and Alliance Size*

We run our experiments on two topologies:

1. a line topology; the transit domains are connected in a line. Each transit domain is connected to 1 to 2 other transit domains. This is shown in Fig. 6.1;
2. a tree topology; the transit domains network forms a tree. Each transit domain is connected to 1 to 3 other transit domains. This is shown in Fig. 6.2.

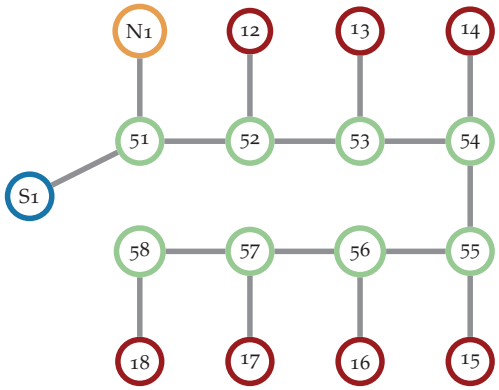


Figure 6.1: line topology

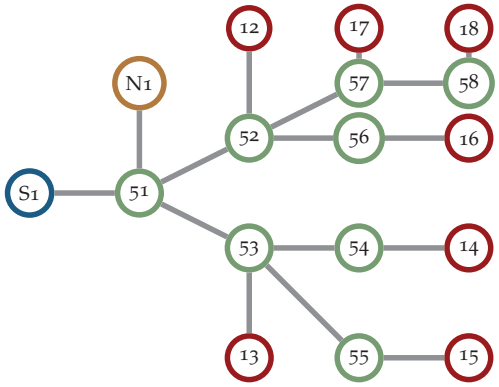


Figure 6.2: tree topology

Topology 1 - line: The topology we use consists of a line of 17 nodes of which (*S1*) is the victim. The transit nodes (*51-58*) form one line, *51* connects to *52* who connects to *53* etc. Each transit domain, except for *51*, has a client domain attached (*12-18*). Client domains interact with *S1* and use a mix of regular and malicious traffic to interact with *S1*. *N1* is an NFV domain; we do not use this domain for the experiments in this chapter.

Topology 2 - tree: The topology consists of a tree of 17 nodes of which (*S1*) is the victim. Transit nodes (*51-58*) connect the clients (*12-18* and are arranged in a tree that expands from *51*. The depth of the tree is 4, with only *58* on that level. Like topology 1: the client domains interact with *S1* using a mix of regular and malicious traffic.

Table 6.1 shows the amount of members, and which members are in the alliance for a given alliance size. The alliance size is based

on the distance of the cooperating domains from *S1*. When size equals 0, there's no cooperation and *S1* acts on its own. When size equals 1, *51* can cooperate in attack mitigation; 2 includes *52* etc. by default, the alliance size includes all transit domains and is set to 8. Note that the amount of members in the alliance is 9 because of the friendly *NFV domain*, *N1*, that is connected to *51*. In topology 2 all transit nodes are already included at size 4, after which increasing the size has no effect on the defence efficiency.

	alliance size	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
line	member count	2	3	4	5	6	7	8	9
	members	N1, 51	N1, 51–52	N1, 51–53	N1, 51–54	N1, 51–55	N1, 51–56	N1, 51–57	N1, 51–58
tree	member count	2	4	8	9	9	9	9	9
	members	N1, 51	N1, 51–53	N1, 51–57	N1, 51–58	N1, 51–58	N1, 51–58	N1, 51–58	N1, 51–58

Table 6.1: The amount of members that are included in the alliance for a given alliance size, for both the tree and line topologies.

6.7.1.2 *Costs and Budget*

Defence costs can differ per domain and cost consists of two components: 1) a fixed component that is always charged when a countermeasure is placed, and 2) a periodic component, an amount that is charged periodically for the amount of time that the countermeasure is active. The total amount of costs that can be spent is bound by the budget.

For our experiments we use equal costs for all domains, we use a value of 0 credits for periodic component and for the fixed component we use a value of 100 credits. Practically, this means that every defensive action costs 100 credits. Therefore, we can limit the maximum budget to 900 credits (100 $\times$ alliance size), which is enough credits to place a countermeasure at all the nodes in the alliance. The default budget for our experiments is 900. We also run experiments for a restricted budget of 300, and 600.

6.7.1.3 *Attack Load*

The attack load is dependent on the amount of attackers and on how much attack traffic they send. We express attack load as the accumulated attack traffic in relation to link capacity on the bottleneck link (the edge between *S1* and *51*). An attack load of 1 is the full capacity of the bottleneck link. An attack load of 2 is twice the capacity of the bottleneck link. We define three load categories low=0.5, medium=0.6 and high=0.9. The default load value for our experiments is set to high or 0.9. We also test for an attack load of 1.0 and 2.0 to see what happens when the attack size exceeds the link capacity.

6.8 RESULTS

We conducted a number of experiments to assess the efficiency for each of the three approaches in Sec. 6.3 on two topologies using the evaluation parameters described in Sec. 6.7.1:

- *Alliance size* (default=8(line), 4(tree)), how far the alliance extends from the victim (Fig. 6.3);
- *Budget* (default=900), the amount of credits that the victim domain has assigned for the complete defence (Fig. 6.4);
- *Attack size* (default=0.9), the accumulated strength of the attack (Fig. 6.5).

We repeated these experiments for a number of scenarios with a varying amount of attackers. We also changed the attackers' position in terms of the number of domains the traffic passes through before reaching the victim. We define *close* attackers as clients connected to the transit domain with the shortest distance from the victim and *far* attackers as clients that send attack traffic via the transit node with the largest distance to victim.

In the end, we selected the following four scenarios, that show distinct attack patterns, in order to highlight the advantages and disadvantages of the approaches:

- *single attacker close*, with the attacker in position 12;
- *single attacker far*, with the attacker in position 18;
- *two attackers, one far and one close*, respectively at 18 and 12;
- *attacks from everywhere*, all clients attack;

For each experiment, we list the efficiency of the defence algorithms on the vertical axis and the values described in Sec. 6.7 on the horizontal axis. Each measurement was repeated 3 times and averaged; the error bars depict the standard deviation from the mean.

6.8.1 *Topology 1 - Line*

Figure 6.3a shows that in single attacker cases Algorithm 3 is performing best, for multi attacker cases Algorithm 4 performs better. In all the cases where one of the attackers is located far away Algorithm 2 performs poorly: for large alliances countermeasures are placed at all the nodes on the path, resulting in higher costs that lower efficiency.

Figure 6.4a shows a similar picture. We can see that Algorithm 2 performs poorly in low-budget conditions. Still, Algorithms 3 and 4 are the best performers where Algorithm 3 is slightly better in single attacker scenarios. In the two attackers scenario, Algorithms 3 and 4 show similar performance.

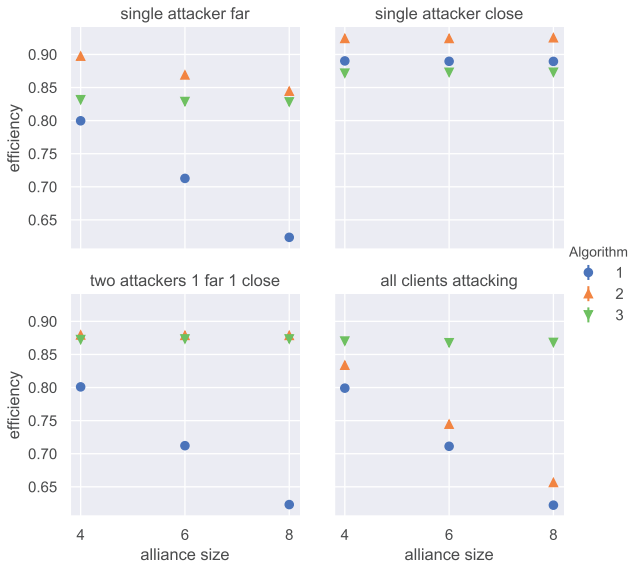
When focusing on the effect of the attack size (see Fig. 6.5a) we observe patterns similar to Figs. 6.3a and 6.4a: Algorithms 3 and 4 perform better than Algorithm 2. However, Algorithm 4 seems to perform better than Algorithm 3 for small attack volumes. Similarly, Algorithm 3 is performing better than Algorithm 4 for larger attack sizes when there's *1 attacker far and 1 attacker close*.

6.8.2 *Topology 2 - Tree*

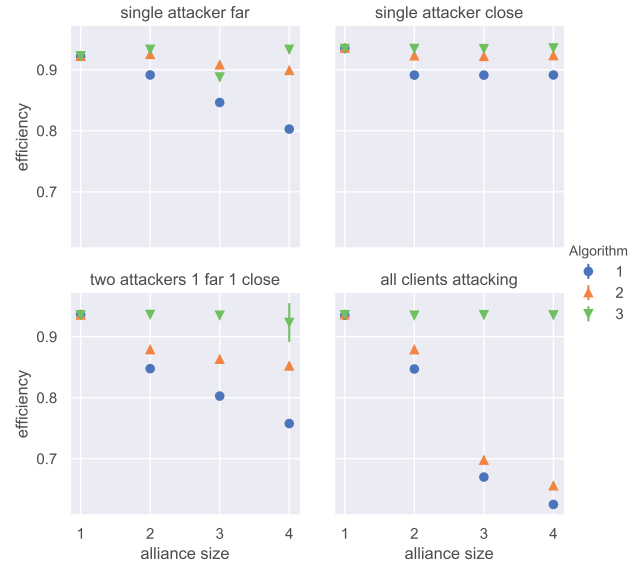
When considering the alliance size in a tree topology, we observe that Algorithm 4 performs better in all cases (See Fig. 6.3b). In the *all clients attacking* scenario Algorithms 2 and 3 decrease more drastically which is due to the tree topology that doubles the amount of members when alliance size increases (see Table 6.1). After $alliancesize = 3$ only a single member is added.

Figure 6.4b shows, similar to Figure 6.4a, that Algorithm 2 is the least efficient approach. In the *all clients attacking* scenario Algorithm 3 is performing as bad as Algorithm 2. In the *all clients attacking* case, Algorithm 3 has to implement the countermeasure at every node in the topology since every node has an adjacent attacker, which is similar to the behaviour of Algorithm 2.

Figure 6.3b compared to Figure 6.3a shows a less aggressive decline for Algorithm 4 in the *all clients attacking* scenario. Again,

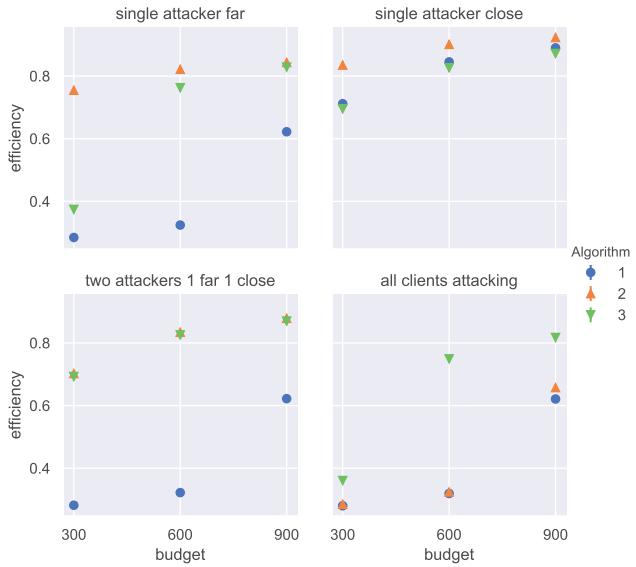


(a) Line, the alliance size grows from 4 - 8

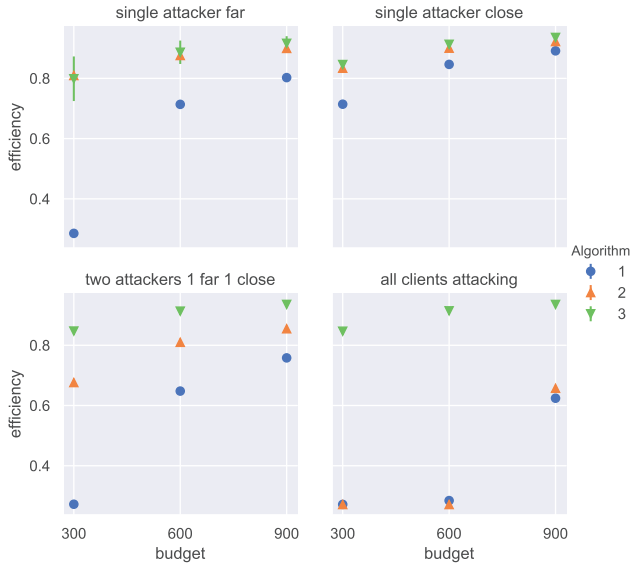


(b) Tree, the alliance size grows from 1 - 4

Figure 6.3: These graphs show how the efficiency of an approach changes in relation to the alliance size. A Higher efficiency is better. Each datapoint is an average of 3 defence attempts; the error bars display the standard deviation from the mean.

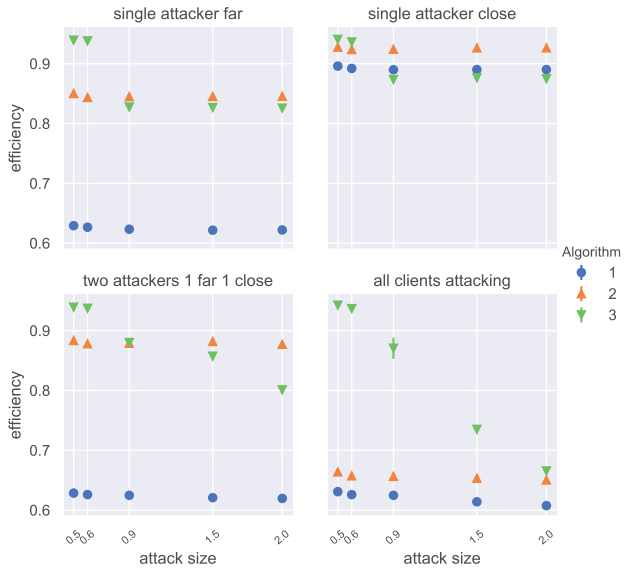


(a) Line

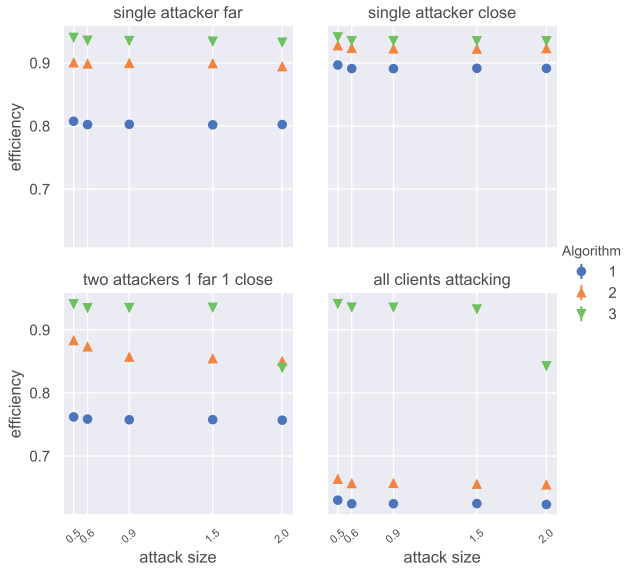


(b) Tree

Figure 6.4: These plots show how the efficiency of each defence approach changes in relation to the budget size. A budget of 900 is sufficient to defend at all nodes. A higher efficiency is better. Each datapoint is an average of 3 defence attempts; the error bars display the standard deviation from the mean.



(a) Line



(b) Tree

Figure 6.5: These graphs show how the efficiency of the approach changes in relation to size of the attack. A Higher efficiency is better. Each data-point is an average of 3 defence attempts; the error bars display the standard deviation from the mean.

Algorithm 4 is the most efficient, except when the attack size is 2.0 in the *1 far 1 close* scenario where Algorithm 3 performs slightly better.

6.9 DISCUSSION

When analysing the results, it is clear that the way a defence is implemented can influence the performance of defending against the attack. We will now highlight the advantages and disadvantages of each approach in terms of efficiency:

- Block everywhere: Algorithm 2 has the lowest efficiency because it is very costly.
- Minimise countermeasures: Algorithm 3 performs well in single attacker scenarios for the *line* topology. This is not the case for the *tree* topology where Algorithm 4 is always more efficient.
- Minimise propagation: Algorithm 4 is the most efficient approach in most cases. The disadvantage is that the approach does not expel the attack from the alliance, the approach only places just enough countermeasures to protect the victim's network. Therefore, the attack traffic still passes through the alliance wasting resources from other members.

There are certain elements that affect the efficiency. First, consider the parameters in the efficiency formula. How α is set in the efficiency formula determines how much emphasis we put on certain factors; changing α will change the efficiency. We set β to .2, a low value, since we compare successfully defended scenarios. Choosing a low β means that attacks that did not recover cannot be more efficient than .2 even if it performs similar when looking at the impacts.

Second, the conditions under which the attack occurs influence the algorithms performance. Algorithms 2 and 3 are directly affected by the alliance size (Fig. 6.3): in Algorithm 2 the alliance size has a direct influence on the cost, since countermeasures are applied at all nodes. In Algorithm 3 the alliance size influences the amount of time it takes to get the required information for all members, before the victim can apply countermeasures. Algorithm 4 is not influenced by the alliance size directly, since the approach stops when the attack is mitigated.

Third, timings influence the performance of the algorithms. The amount of time it takes to request information can negatively affect performance. In Algorithm 3 the algorithm first collects information for all nodes. If one of the nodes takes a long time to respond, the algorithm cannot continue until the data is received. This delay decreases efficiency. The response time affects Algorithms 2 and 4 to a lesser degree, since they apply countermeasures on a per node basis. As mentioned in Sec. 6.3, Algorithm 4 is dependent on the *wait time*, which is based on the time it takes to detect recovery. We used a trial and error method to obtain those values for our system. In our case the *wait time* is set to the amount of seconds it takes to collect enough samples of our metrics to detect recovery: 16s, plus a 3s margin for the state to propagate through the system. Setting *wait time* to a large value increases the impact of the attack and makes Algorithm 4 less efficient.

Finally, all algorithms are affected by budget constraints as can be seen in (Fig. 6.4).

Heuristics such as using knowledge of existing topology information can improve the current algorithms. The first time one of the algorithms is executed it can also construct a view of the network topology, which can be used, subsequently, to shorten query times. For example, an algorithm can directly ask the border domains in the alliance to block incoming attack traffic, or an algorithm can first target well connected domains in the alliance to increase the effect of the first blocking action. When using additional data sources, the information should be kept up to date for optimal results.

When starting our experiments, we expected that the *Minimise Countermeasure* approach (Algorithm 3) would be the most efficient under budget constraints. This approach blocks the attacks at the source and is considered to be an effective approach against DDoS attacks [60, 72]. Instead, we found that the *Minimise Propagation* approach (Algorithm 4) was more efficient. *Minimise Propagation* mitigates attacks sooner because this approach does not have to trace the attack back to the attackers. The approach also works with fewer countermeasures because it stops when the attack impact is sufficiently reduced. Both factors influence efficiency positively.

However, *Minimise Countermeasures* may be more efficient if defence costs are proportional to the amount of attack traffic: At the border of the alliance, there is only a low amount of traffic. The amount of traffic accumulates closer to the victim. *Minimise propagation* becomes less efficient the moment that these proportional costs are included, as it defends close to the victim where the attack traffic volume is highest.

Finally, there may be other incentives for defending close to the attack source, such as reducing attack footprint throughout the alliance. If these incentives can be quantified, they can be used as an additional input parameter to efficiency. Also in this case, we expect that *Minimise Countermeasure* would be the most efficient approach.

Minimise Countermeasure does not defend against potential attackers that are in the alliance. Although the other two approaches are usable against this type of attackers, this approach is not because it implements the countermeasures at the alliance border instead of close to the victim. Since this work only focuses on attackers outside of the alliance, it remains to be verified how efficient the approaches are against attacks that originate from inside the alliance.

6.10 RELATED WORK

A significant amount of research can be found on collaborative DDoS detection and response. Most of the papers focus on multi-domain detection [17, 18, 105, 118], multi domain defence [99, 100] and combinations of the two [45]. All of these articles focus on DDoS attacks specifically. We use DDoS response only as a basic use case, yet the approaches that we evaluate in this chapter can be used for asking assistance from friendly domains for *any* attack pattern. The collaborative defence approaches in this chapter can be used as a base for defending against other attacks.

Our work presumes that all participating domains in an alliance are willing to cooperate. In principle one would want to verify that the other parties are really acting upon each request. Mannhart et al. [67] discusses four approaches to ensure effectiveness of a cooperative mitigation. The work focuses on validating that the "mitigator" correctly applied the mitigation. The authors pursue tamper-proof execution and verification of an applied countermeasure. They conclude that none of their four approaches alone is capable of providing this. Since we expect that the collaboration happens in an alliance context, there is a basic level of trust present between the members including rules and regulations on how to handle other members' data. The evaluation of the defence happens from the victim's perspective and does not rely on any guarantees from its collaborators.

Meng et al. wrote a taxonomy on collaborative security systems [71]. They assess multiple security related systems that use both collaborative detection and collaborative response to achieve their goals. They identify a number of challenges in these types of

collaborative systems. Our work addresses a number of them: in regard to privacy we adopt a limited information sharing mechanism (see Sec. 6.2.1); incentive is in our case implicit in the existence of the security alliance; when looking at robustness we avoid a single point of failure by not relying on a central coordinator.

6.11 CONCLUSION

In this chapter we have evaluated three different approaches to defending against attacks in multi-domain settings. To do this, we introduced a generalised formula for efficiency, that can use any finite number of parameters (multiple impacts, costs). Our work shows that the efficiency of a multi-domain defence depends on the order of asking the members to act and the location of the asked member relative to the attackers.

Furthermore, our results showed that *Minimise Propagation* (Algorithm 4) is the most efficient approach to take when defending against a DDoS attack. Besides the order, and location, system timings are another important factor: if tasks are dependent on each other and the first task takes a long time to complete, the second task has to wait. This increases the overall run-time of the defence, which impacts its efficiency. Waiting for the system to determine recovery also negatively impacts the defence efficiency.

Our future work will focus on evaluating a fourth approach that uses the Social Computational Trust Model described in [27]. The work claims that alliance members would rather ask for support from members that showed that they are capable and willing to help (*evidence-based trust*). We want to assess if such an approach based on these trust values shows better efficiency than the ones evaluated here.

Another research direction involves network topologies. VNET allows us to instantiate any topology on which we can run our attack scenarios. In this chapter we used two basic topologies: *tree* and *line*. We plan to further evaluate the efficiency of our approaches on topologies that are used in practice by Internet Service Providers.

Finally, it is interesting to research how dynamic costs affect efficiency. In this chapter we did not use periodic or dynamic costs; we also kept the costs equal at all collaborators. Introducing dynamic budgets based on attack traffic or on periodic cost, for as long as the countermeasure is active, has an effect on efficiency. Due to the dynamic cost, defence costs are not known a-priory and since

active defences consume budget over time one may have to switch to another defence approach halfway in order to remain efficient.

7

MULTI-DOMAIN TRUST-BASED COLLABORATION FOR DEFENDING AGAINST ATTACKS

This chapter demonstrates how using trust, based on evidence that is gathered over time, improves the efficiency of collaborative defences. We use the Social Computational Trust Model developed by Deljoo et. al. [27] to implement trust in multi-domain alliance. We extended the SARNET-agent by adding the components that are needed for gathering the evidence that we use to compute trust and risk. Using these trust components, we create a defence that computes the risk of collaborating with each member, and asks the members with the lowest risks to execute the defence tasks first. We compare the trust-based defence approach to the defence approaches in Chapter 6, in an alliance where not all members are equally cooperative. This chapter contributes to the answer to RQ3 by identifying trust as an important factor in multi-domain defences when not all members are equally cooperative.

This chapter is based on:

- **R. Koning**, A. Deljoo, L. Meijer, C. de Laat, and P. Grosso *"Trust-based Collaborative Defences in Multi Network Alliances"* [50], in *2019 3rd Cyber Security in Networking Conference (CSNet) (CSNet'19) [accepted]*, © IEEE.
- A. Deljoo, **R. Koning**, T. van Engers, L. Gommans, and C. de Laat *"Managing Effective Collaboration in Cyber-security Alliances Using Social Computational Trust"* [26], in *2019 3rd Cyber Security in Networking Conference (CSNet) (CSNet'19) [accepted]*, © IEEE.

7.1 INTRODUCTION

Multi-domain collaborations can be organised using alliances, in which members agree to a common set of rules to establish a base of trust [28]. Yet, even with an alliance in place, based on the ability, the willingness to help, and experience, some members will be more trusted than others. This trust differs per member and is based on the members' experience and the collected evidence about the member in question.

In Chapter 6 we showed that the efficiency of collaborative defences, even when applying the same task, can differ based on the

approach and order in which the collaborators are asked for help. Since asking help from more trusted members may yield better results in defending, we ask ourselves: If we have an indicator of trust, can this be used in ally selection and how does trust affect the efficiency of collaborative defences?

7.2 MULTI-DOMAIN SARNET

SARNET is a framework for detection and autonomous mitigation of attacks on computer infrastructures. When multiple defences are available for the situation, the SARNET picks the *defence* with the highest efficiency ranking and executes it. A defence consists of multiple *tasks*. These tasks can be performed by the domain itself or be delegated to collaborators. Collaborative defences require each participating domain to run its own SARNET agent. The agents are responsible for coordinating activities between collaborators in the alliance. These activities include coordinated responses to threats as well as sharing information such as threat intelligence or analytics data with agents of other domains.

In multi-domain defences we distinguish four categories of requests:

- Simple informational requests: Requests that can be answered directly based on the knowledge of the requested member.
- Complex informational requests: 1) Request for information that is provided over time at the discretion of the requested member i.e. a request that subscribes to an information service, 2) Information that is collected (from other sources or members) and transformed by the requested member.
- Simple actionable requests: The decisions and actions are driven by the requester. Simple actionable requests are executed directly by the requested member according to the requester specification.
- Complex actionable requests: The decisions and actions are driven by the requested party. 1) Automatic mitigation services allow the requested member to apply certain countermeasures at their discretion at some point in the future. 2) Delegated actions allow the requested member to further handle the orchestration the multi-domain defence.

Both types of simple requests consist of actionable tasks or queries and the performance can easily be verified. For an actionable task

one can measure if the desired effect took place and the informational query is shortly followed up with a response. Complex requests often consist of multiple sub-tasks, or queries to other members. These tasks usually take longer to complete or generate multiple replies over an extended time period. The quality of the responses depends on the amount of effort the requested domain is willing to spend on the analysis, the execution, or the processing. Therefore, it is important that the remote domain acts in the interest of the requester.

We implement SARNET on top of our platform discussed in Chapter 6.

7.2.1 *Attack Scenario*

Using VNET we can construct a virtual infrastructure by supplying a topology, (Fig. 7.1), that uses virtual machines which represent the following domains:

- a *service domain* (V) contains a web service that resembles a marketplace where clients make purchases;
- a *transit domain* (51–58) forwards traffic, it provides basic blocking, redirection and rate limiting functions;
- a *client domain* (12–18) interacts with the service domain by making transactions with the service domain;
- a *NFV domain* (61) provides a set of network functions (using Network Function Virtualisation, NFV) that can be used for further analysis, such as an Intrusion Detection System, or a honeypot, or can be used as a countermeasure such as a traffic scrubbing NFV.

Traffic flows back and forth from the client domains via the transit domains between the service domain. Normally, the traffic consists of transactions (simulated purchases) sent to the service domain. When we start an attack, we instruct one or more client domains to attack the victim, which is the service domain. The attacks consist of UDP based (Distributed) Denial of Service attacks initiated from one or more client domains (12–18) that congest the link between the victim (V) and its upstream provider transit domain (51). The defence consists of filtering out this traffic pattern at the selected members in the alliance. The alliance consists of all the transit domains, the service domain, and the NFV domain and excludes the client domains.

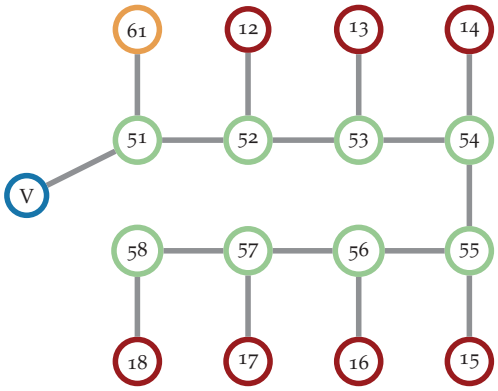


Figure 7.1: The network topology used in the experiments, V is the victim, 12–18 are attackers, 51–58 map to transit51-58 and are transit domains, 61 is nfv61 the NFV domain.

7.3 TRUST IN SECURITY ALLIANCES

Trust is considered one of the success factors for alliances [23]. To form a security alliance, trust among the members needs to be organised, maintained, and measured. Deljoo et al. [30] define trust in the alliance as follows: "a trustor expects a trustee to perform task t and the trustee will not exploit vulnerabilities of the trustor when the trustee is faced with the opportunity to do so." Therefore, every trustee:

- has an ability to perform task t (competence),
- fulfils the commitments towards the trustor (integrity), and
- acts towards the trustors interest (benevolence).

In order to organise, maintain, and measure trust among the members, we propose to use the social computational trust model (SCTM) as described in [27]. The SCTM model evaluates the trust of a trustee based on the three distinctive factors: benevolence, competence, and integrity; this work assumes that integrity is provided, automatically and fully, by being a member of the alliance. Therefore, this work does not include the computation of integrity.

7.3.1 *Notation*

Let us denote alliances members as A where $l, r \in A$. The interaction history is stored in the evidence knowledge base E_{Kb_l} and is

called evidence. We assign three different values to the evidence using function $val(E)$: 1, when *FD* (fulfilled), .5 when *FDD* (fulfilled with delay), and 0, when *V* (violation/not fulfilled). The E_d function obtains direct evidence from the E_{Kb_l}. The $E_{in(t)}$ function extracts indirect evidence from all $E_{Kb_{r_{nbr}}}$ for task t where nbr are the neighbours of r. We initialise the knowledge base for the topology in Fig. 7.1, using the values in Table 7.3.

7.3.2 *Benevolence*

Benevolence is considered as one of the key trust components and the antecedent of trustworthiness (e.g. [57, 63]). The benevolence value is derived from the mutual interaction between a trustor and a trustee. According to [27] the benevolence of trustee r towards trustor l is computed by:

$$Ben(l,r) = \frac{1}{|N|} \sum_{l,r \in A} (val(E_d(l,r;E_{Kb_l}))), \tag{7.1}$$

where N is the number of entries in E_{Kb_l} with the defined value, in which l has interacted with r.

7.3.3 *Competence*

Competence refers to the ability of the trustee to perform the task. We assume that l and r have not collaborated before. Therefore, the trustor needs to request the evidence form the trustees' direct neighbours (r_{nbr}). The competence of node r as the trustee is given by:

$$Comp(l,r,t) = \frac{1}{|r_{nbr}|} \sum_{l,r \in A} (val(E_{in(t)}(r_{nbr},r;E_{Kb_{r_{nbr}}}))), \tag{7.2}$$

where $|r_{nbr}|$ is the number of neighbours who respond to the request of r. The benevolence and competence functions return a value between [0,1].

7.3.4 *Risk*

Every collaboration comes with a risk that needs to be minimised. Das et al. [23] defined the relational and performance risks as two distinct risks for the alliance, and Deljoo et al. [30] present the risk evaluation framework for the security alliance. Relational

risk concerns the behaviour of the alliance members, $1 - Ben(l, r)$, while the performance risk considers the competence of members to perform the given task, $1 - Comp(l, r, t)$. The risk is calculated using an updated version of the equation defined in [30], Section VI:

$$Ri(l, r, t) = \alpha(1 - Ben(l, r) + (1 - \alpha)(1 - Comp(l, r, t)) \quad (7.3)$$

In Sec. 7.2 we mentioned that for complex messages it is important that the requested domain acts in the interest of the requester i.e. have a low relational risk. When the relational risk is considered more important, α (a value between $[0, 1]$) can be set to a higher value than the default of .5.

7.4 TRUST BASED DEFENCE PREREQUISITES

To implement trust based defences we need to extend the existing SARNET agents. Since the trust is based on the time it takes for an ally to perform an action we need a message tracking mechanism. In Sec. 7.4.1 we explain how we use this tracking mechanism to gather the evidence required for the trust computation. In Sec. 7.4.2 we show how we compute the trust values that are used in the trust based defence algorithm in Sec. 7.5.

7.4.1 *Message Tracker*

Since the building of evidence is based on fulfilling requests within a time period, we need to implement a method of time tracking the messages. VNET uses an asynchronous programming model; sending and responding is handled in different parts of the code and the response handler has no information on what is being sent. For time tracking to work, we need to map the response back to the request. Therefore, we modified the original message format to include a message ID and a transaction ID which we also store upon sending. The recipient can now reply with the message ID included such that the sender can look up the corresponding request in its *track database*, a database that keeps track of outgoing requests. Table 7.1 shows the format of entries in the *track database*.

When a message is sent we store the message ID, the message, the message type, the sender, and the receiver in the *track database*. Additionally, we set the *t_req* field to the time the message is sent.

To each message sent we expect two replies: 1) an acknowledgement that the message is received, and 2) a message that the task

is completed whereby each reply includes the message ID. When data is requested from the recipient, the recipient returns the data in a third message that includes the transaction ID. When the acknowledgement or the task completed messages are received by the sender, the sender will store the current time in *t_repl* in case of an acknowledgement, and *t_done* when the task is completed.

Periodically[1], we prune the *track database*. We convert the completed and expired transactions in the *track database* to evidence and store them in the evidence knowledge base (evdb). When *t_done* is set for this request we store the difference between *t_req* and *t_done* in the evdb. When *t_done* is not set and the difference between *t_req* and the current time exceeds the timeout, the request did not complete in time and we store 0 in the evdb.

The *track database* only contains messages that are "in flight"; when no messages are exchanged and the timeout has expired the database should be empty.

msg_id	msg	type	local	remote	t_req	t_repl	t_done

Table 7.1: Track database entry format: we store the id of the message, the complete message, the message type, local and remote agent identifiers, the time when the request is made (t_req), the time on which we receive the acknowledgement (t_repl), and the time we received the task done notifications (t_done).

7.4.2 *Trust Computation*

When computing trust, risk or any of their components, we consult the evdb. The format of the evdb is described in Table 7.2. We store the task itself, the trustor (source), the trustee(destination), the time that has elapsed to complete the request, the request id, and the time when the message is recorded. When the trust values are requested, we use the algorithm from [29] to convert the elapsed times to evidence.

Benevolence is based on all the fulfilled evidence in the evdb. To compute benevolence, we require the name of the remote agent. When the function is given the optional argument *time*, the function

[1] The function should be called approximately every second but the timing can vary due to other periodic functions being executed in the same thread.

only computes benevolence using the evidence that is produced after that time. When the evidence is obtained we calculate benevolence according to Eq. (7.1). If the benevolence computation fails, the benevolence implementation returns "None"

Competence depends on data acquired from all the neighbours of the remote agent. Computing competence requires the task and the target and, like benevolence, the function returns "None" when the value cannot be computed. To acquire the data from member domains, competence relies on the following (simple informational) requests:

- Trust Request: Requests information for some trust component. For *competence* the request must also contain the task in question.
- Trust Response: Responds to the trust request by providing the requested evidence of the member specified in the request.

When the relevant evidence is acquired, we compute the competence according to Eq. (7.2). Acquiring competence is an expensive operation where the requester has to wait until all the neighbours of the target member reply with their evidence. A neighbour can take some time to reply because the request has to be sent out on the network; a relatively slow medium which is subject to delays, and round trip times. Since competence can be called for multiple times per second and the reply is not instant, the long running requests will be sent out multiple times per second, consuming resources and congesting the network. To prevent congestion from happening, we cache the result from competence for 5 seconds. Every subsequent request is now answered from the cache until the cache is cleared. When there is no answer in the cache, the requests will be sent over the network to obtain the updated values. Caching introduces a trade-off between the latest data and performance, but it is necessary to prevent excessive use of the network. For the caching to be effective, we recommend to keep the time interval that clears the cache larger than the response time. In our case we clear the cache every 5 seconds, which is larger than the response time of the agents on the network (in our case < 1 second).

Risk is provided by taking the benevolence and accumulated competence values and computing them according to Eq. (7.3). If the value cannot be computed, the function returns "None". Since the tasks that are executed only consist of the simple request types (See Sec. 7.2) we do not prioritise relational risk over performance risk. Therefore, we calculate risk using: $\alpha = .5$.

task	source	destination	elapsed	id	time

Table 7.2: Evidence knowledge base entry format: A single entry stores the task (in our case message type), the source and destination agent, the elapsed time of the request (0 if the task never completed), the id of the message, and the time when the evidence is gathered.

7.5 RISK BASED DEFENCE IMPLEMENTATION

Algorithm 5 shows how we implemented the risk-based approach. The algorithm has two distinguishable phases: 1) Lines 1–6. When the defence starts we first collect the necessary values: risk and benevolence, to rank the nodes.

Once these values are collected, they will not be updated as long as the defence is active. To break ties when two nodes have equal risk and benevolence, we assign a third ranking value to a random and unique number. In our experiments, we seed the random number generator, in order to achieve consistent results. 2) Lines 7–13. When the trust information is gathered and the ranking is complete, the actual defence starts. The top ranked domain is asked whether or not it detects the traffic and to deploy a defence when this is the case. To each queried member it will exchange the following messages:

- **Ask**: Asks a member whether or not it has seen traffic according to a certain pattern and from which of its neighbours the traffic originates.
- **Match**: The response to the ask message. This message contains the responding domain's neighbours on which the pattern is seen.
- **Deploy**: Deploys a filtering action on the specified traffic pattern towards one of the members' neighbouring domains.

Ask is considered a *simple informational request* and *deploy* is considered a *simple actionable request*

7.6 EVALUATION

First, we need to verify whether or not the evidence collection and the trust based defence algorithm behave as expected. To achieve this, we attack the victim using the same attack multiple times, and

Algorithm 5: Risk Based Algorithm

Input: *pattern*: attack pattern
alliance: alliance members
time: wait time
for $n \in alliance$ **do**
$Risk_n \leftarrow$ gather_risks(task, n);
$Benevolence_n \leftarrow$ gather_benevolence(n);
$Random_n \leftarrow$ random();
end
ranked $\leftarrow$ sort N on *Risk*, then on $1 - Benevolence$, and then on *Random*;
for *node* $\in$ *ranked* **do**
if *attack not resolved* **then**
ask node for its *neighbours* that produce *pattern*;
deploy countermeasure at *node*;
wait for *time* seconds;
end
end

between each time, we make sure that the network returns to a normal state. After receiving the attack and implementing defensive actions, the victim should have more evidence available about the other alliance members. The trust values that are computed using the new evidence should allow the victim to make a better decision on which members to ask for help while defending. To validate, we first set the initial evidence such that it mismatches the behaviour of the nodes; this mismatch should immediately result in unfulfilled tasks in the evidence database and in turn change the ranking for the next defence. In Table 7.3 we list how we initialise the network. We give a preference to transit51 by initialising with two extra fulfilled (FD) transactions. We demote the use of nfv61 by adding two extra violated (V) transactions.

To see how the evidence changes the trust values over time, we set members to behave differently from the initialisation (see Table 7.4). Transit51 is initialised as more trustworthy and is given a low success rate, which should result in to violations when being asked. The success rate of each member is listed in table Table 7.4.

This means we expect two important things to happen: 1) In the first attempt to defend, transit51 is considered more trustworthy and gets selected. During that attempt, it will generate many violations (V). Therefore, we expect that transit51 should move down

node	FD	FDD	V
transit51	3	1	1
transit52	1	1	1
transit53	1	1	1
transit54	1	1	1
transit55	1	1	1
transit56	1	1	1
transit57	1	1	1
transit58	1	1	1
nfv61	1	1	3

Table 7.3: Initial evidence in the evdb: the table shows the amount of fulfilled (FD), fulfilled with delay (FDD) and violated (V) requests for each of the members participating in the alliance.

to the bottom. 2) Transit52 has a high success rate which results in fulfilled requests (FD). Because of the the high success rate, we expect transit52 to end up on top of the ranking. The other nodes are expected to converge to the rank according to their success rates.

We will now execute the attack 10 times and display how the ranking changes: Secondly, we need to see the effect of the ranking on the efficiency.

Finally, we compare the efficiency of this approach to the efficiency of three original approaches we evaluated before in Sec. 6.7:

- Approach 1 - Counteract Everywhere; places a defence on every ally that sees attack traffic starting near the victim and working its way to the attackers.
- Approach 2 - Minimise Countermeasures; discovers where the attackers are located by recursively asking the allies from where the attack originates, and then defends the attack close to the attackers at the border.
- Approach 3 - Minimise Propagation; similar to Approach 1, this approach places a defence, starting at the ally closest to the victim. Only this time we wait for a time period to notice the effect. When still under attack the approach continues to ask the members' neighbours to defend.

node	success rate
transit51	0.0000001
transit52	1.0
transit53	0.8
transit54	0.6
transit55	0.4
transit56	0.2
transit57	0.1
transit58	0.0000001
nfv61	1.0

Table 7.4: The member behaviour for our experiments: the table lists the probability of successfully executing a task for each of the members in the alliance.

7.7 RESULTS

The topology used for the experiments is listed in Fig. 7.1. We use the same four scenarios as in Sec. 6.3 to compare the results; each scenario portrays distinct attack conditions:

- *single attacker near*, with the attacker in position 12;
- *single attacker far*, with the attacker in position 18;
- *two attackers (1 far, 1 close), one far and one close*, respectively at 18 and 12;
- *all clients attacking*, all clients attack;

Figure 7.2 shows how the ranking of each node evolves over time. During each attempt to defend, the victim gathers more evidence on the alliance members. The next time we defend, we recalculate the trust ranking based on the newly collected evidence. The rankings are snapshots of a single run; they are not averaged. Each line represents a member; we take snapshots of the ranking at the moment the defence is created. As expected, the first attempt ranks the members according to the initialisation. For the subsequent attempts the ranking changes until it converges to the set member behaviour and stabilises as expected.

Figure 7.3 shows how the efficiency [81] of the trust based approach changes as the victim gathers evidence. Each data point is an

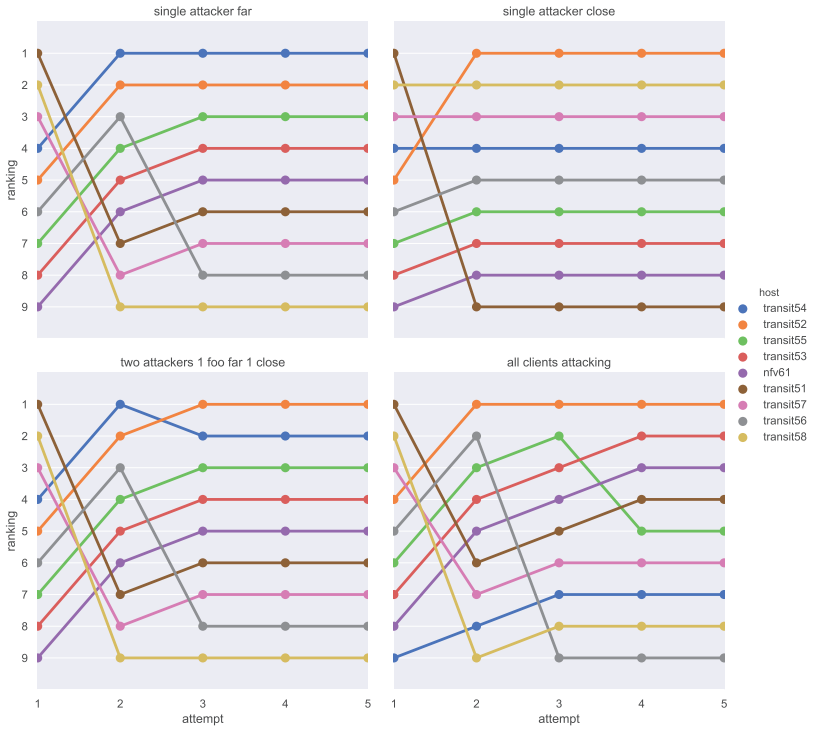


Figure 7.2: This plot shows the changes in the ranking of the members after defending x times. The ranking changes when the trust numbers are updated based on the gathered evidence. The highest ranked member (1) gets selected first by the victim to place a countermeasure.

average of three runs, the error bars contain the standard deviation from the mean. There is not much variation between the runs, so the error bars in the plots are small. It is clearly visible that in the first run, when transit51 is highly ranked (see Table 7.3), transit51's efficiency is lower than its efficiency in the consecutive runs. Also in all cases the efficiency seems to increase and stabilise over time. The variations after the stabilisation can be explained because we are working with probabilities driven by random numbers that change over time.

Figure 7.4 shows the trust-based approach we discussed in this chapter, in comparison to the approaches we studied in Chapter 6. On the x axis *attempt* indicates the defence attempt using the same evdb. The different colours or symbols indicate the different approaches. The trust based approach is indicated as *Algorithm 4*.

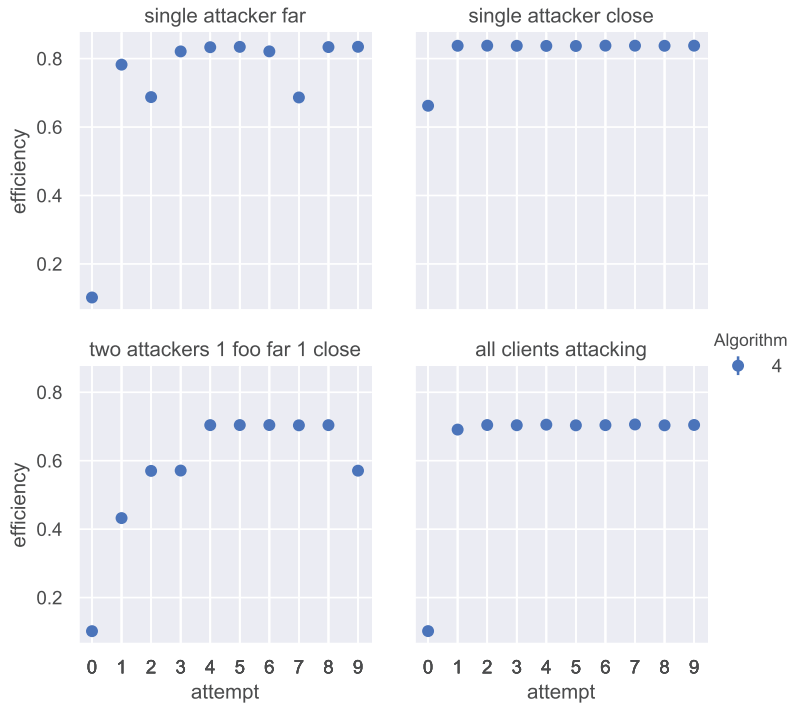


Figure 7.3: Changes in efficiency after defending x times

In Figs. 7.4 and 7.5 we compare the approaches under two different budget constraints: high (unlimited) budget, and low budget. Each member charges 100 credits to implement a countermeasure. Thus, with a budget of 900, the victim can ask all the members of the alliance to implement a countermeasure. With a budget of 300 (Fig. 7.5) the victim can, therefore, only ask 3 members to implement a countermeasure each defence attempt.

When the budget is large enough to ask all the members (Fig. 7.4), the rankings converge better than when the budget is limited. For the first attempt, the trust-based approach is less efficient than the other approaches, but from the second attempt onwards, the graphs show efficiencies close to the other approaches. In the *single attacker far* and the *all clients attacking approach* the trust-based approach (Algorithm 4) is more efficient than the others.

In the constrained budget approach (Fig. 7.5), we can see that the trust-based approach (Algorithm 4) is not performing optimally. Still, in the single attacker far case, it performs better than the other algorithms. For the other cases, we can see that Algorithm 4 converges to higher efficiency, although the efficiency converges

slower than in (Fig. 7.4). The slower convergence of the rankings can be explained by the fact that the algorithm can only send requests to the top three nodes in the list and thus only collects new evidence from the three (highest ranked) members in the alliance.

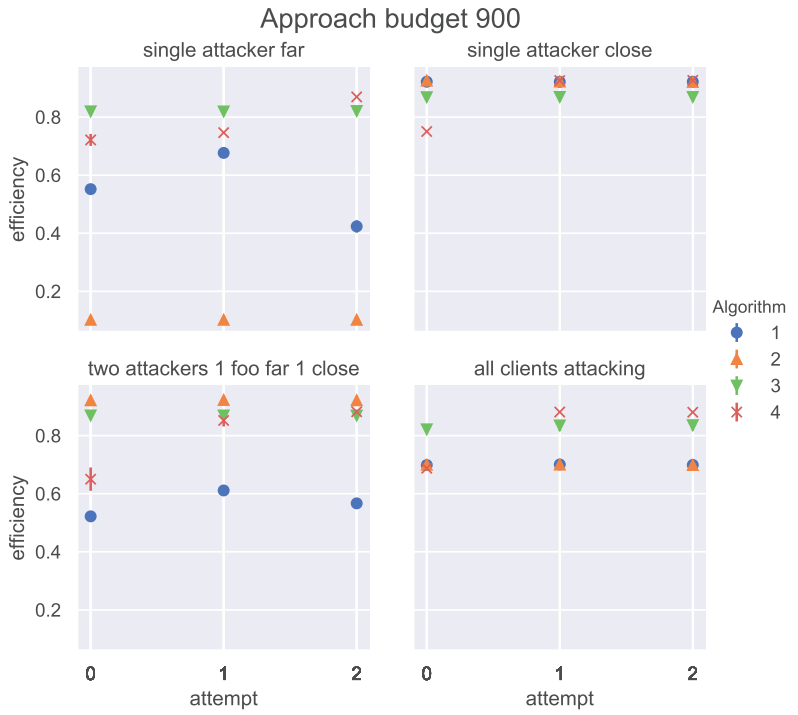


Figure 7.4: Comparison of the trust-based approach (Algorithm 4) to approaches from Sec. 6.3 with unlimited budget

7.8 DISCUSSION

When the defence budget is constrained, we noticed some side-effects while using the trust-based approach. Due to the constrained budget, we can only ask a limited amount of members for help during each defence attempt. Because we rank the members according to trust, these are only the most trusted members. When a lower ranked member changes behaviour and suddenly becomes benevolent and competent, it will never move up the ranking, because

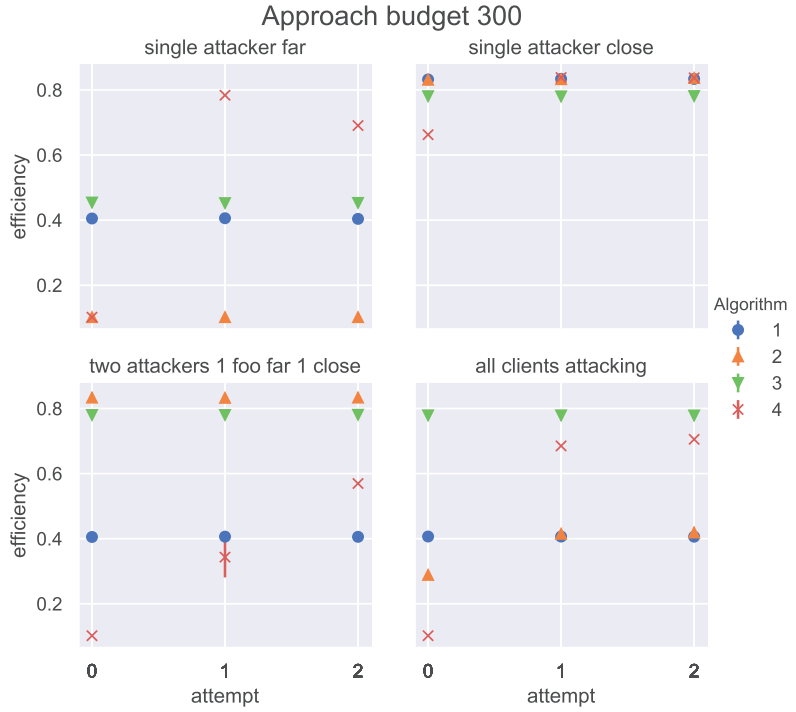


Figure 7.5: Comparison of the trust-based approach (Algorithm 4) to approaches from Sec. 6.3 with very constrained budget

of the lack of interaction and evidence. Only when highly ranked members move down in the ranking, the evidence for some of the lower ranked members are collected: this could cause a low ranked and (suddenly) trustworthy member to move up. This side effect can be seen in Fig. 7.5, where we constrain the budget, and Approach 4 converges slowly and never reaches the same efficiencies as in (Fig. 7.4).

One prerequisite for the trust-based defence, the message tracking implementation, adds significant overhead to the system. While originally receiving zero replies, with the message tracking implementation enabled, a single message can now receive up to three replies: the acknowledgement, the task done message, and the actual reply. When under heavy attack, these messages increase the load of the system, and can also contribute to the congestion problem that is already caused by the attack. Removing the acknowledgement messages, which are currently not used by the trust-based algorithm, and merging the task done message into the actual reply could help to reduce this overhead.

Our trust-based approach can be further improved using heuristics. Currently, our trust-based approach asks the most trusted members in the alliance for help, regardless of whether or not the member is transporting attack traffic. If we establish an attack tree first by asking neighbours from which of their neighbours the attack originates, recursively, we know exactly which members transport attack traffic. With the initial (time) penalty of learning the attack tree, can limit the trust computation and member selection to a list of members that actually forward the attack traffic and, hopefully, be more efficient in the overall defence.

A trust-based approach can also introduce new attack vectors. Since the victim always send its requests to its most trusted allies first, the amount of communication between them may be higher than with the members who are less trusted. Even when this communication is encrypted, by observing network communication patterns, a smart attacker may be able to identify the victims most trusted allies. The attacker can use this information to gain an advantage in bringing down the victim by neutralising the victims allies first. If agents communicate over public infrastructure this information leakage is a potential risk. Establishing and using an encrypted peer-to-peer overlay network for agent communication can obfuscate these communication patterns: The victim can send the messages to any alliance member and the peer-to-peer overlay will deliver the messages to the target agent.

7.9 RELATED WORK

Shabut et. al. describe a recommendation-based trust model with an effective defence scheme for mobile ad hoc networks [93]. Their focus is mainly on preventing bad or malicious recommendations from nodes by providing defences against attacks on the trust model. The solution they propose relies on three centralised components, on which they perform statistical analysis and a clustering technique to detect deviations in trust and to prevent malicious recommendations. Our work focuses on how trust can be used to improve defences on the network infrastructure. We recognise there are also attacks that target the trust infrastructure, these attacks are beyond the scope of this thesis.

Chen et. al [17] discuss a collaborative multi-domain detection system for DDoS attacks. The authors developed a secure infrastructure protocol (SIP) to establish mutual trust or consensus. In SIP they adopt the adaptive trust negotiation and access control

framework[90] which uses the PeerTrust[74] trust management system to establish trust. The trust used in these systems is established by matching credentials of the requesting party to access lists and policies defined by the responding party. Our work uses a different form of establishing trust. The trust we use is based on the behaviour of the participating nodes and their attitude towards benevolence, competence, and, integrity.

7.10 CONCLUSION AND FUTURE WORK

Using trust as a criterion for partner selection in multi-domain collaborative defences seems promising. Maintaining a short list of members who are responsive, capable, and willing to help can be beneficial when defending collaboratively. The results show that the trust-based algorithm increases in efficiency over time as more evidence is gathered from the members in the alliance. In cases where the budget is limited, and the collected evidence only comes from a part of the alliance, the efficiency converges slowly and remains lower than when using the original approaches. At full budget, the trust-based approach converges faster to comparable, and in some cases even higher, efficiencies than the original approaches. In general, we can conclude that there are benefits when considering the members' trust in selecting candidates for task placement in multi-domain collaborative defences.

We propose to increase efficiency further by first establishing an attack tree and, consecutively, applying the trust-based response on the nodes in the tree. More research is needed to see whether or not the impact of establishing the attack-tree outweighs the benefit of having a more focused list of members that can be asked for help. For now, we assume that integrity is considered a condition for joining the alliance, all members of the alliance are considered to have the same integrity. In the future we intend to include integrity in the risk calculation. We also pointed out that unintended information leakage can occur when depending on the most trusted allies. Researching if and under which conditions trust-based communication patterns leak trust information and how to mitigate this leakage will be a necessary continuation of this work.

8

CONCLUSION

As attacks increase in frequency and complexity, it becomes critical that security professionals can focus on the few important attacks that cause considerable damage and on discovering and defending against new threats. Networks that autonomously defend against attacks are a critical step in network automation and service protection. The automatic defence against known attacks ultimately allows security engineers to focus on identifying new threats and developing software to defend against new attacks. In this thesis, we show that recent software developments, especially in the fields of SDN and cloud technologies, allow us, as shown via the prototypes described in this thesis, to build networks that can autonomously respond to cyber attacks. By the exploration and development of the SARNET prototypes in this thesis, and by examining the results and experiences we gained from working with these prototypes in both single and multi-domain environments, we identified the strong points and the limitations of the technologies used and obtained the insight that allows us to define the steps, factors, and patterns that are required to construct autonomous defences in practice.

The main contributions described in this work are:

- a conceptual framework for autonomous response,
- a software defined experimental platform to evaluate defences,
- a control framework for SDN and NFV based defences,
- metrics for defence performance evaluation,
- a toolkit for gathering contextual information about attacks by cross-referencing data sources.
- a social trust-based multi-domain defence approach.

Using these contributions and the insights we gathered during the development and implementation of the frameworks and the proof of concepts, we will now be able to answer the research questions.

8.1 ANSWERS TO THE RESEARCH QUESTIONS

Before answering the main research question we first answer the sub questions.

- RQ1: *How can we determine the security state of a network?*

We determine the security state using the following procedure: 1) define observables that qualify whether or not metrics of the system, or the network, are within the range of normal operation, 2) monitor changes of these observables, and 3) classify attacks by observing changes in specific combinations of these observables.

Throughout this work we visualise (Appendix A) the security state and attack impact using the methods described in Chapter 3. We found that using the procedure above we can detect attacks, classify them, and start defences automatically, according to the classification (Chapter 4). In practice there are more data-sources that can provide information about attacks. In Chapter 5 we describe a framework that can cross-reference the data provided by observables to other data sources in the network and gather a more complete view of the situation. The contextual information that is gained from this process can be used to complement and improve the attack classification which results in a more detailed security state.

- RQ2: *How can new developments in computer network control contribute in creating countermeasures to attacks?*

Our research shows that network developments can contribute in two ways:

1. In Chapter 3 we identified some of the basic actions that an SDN provides that are useful to defend against attacks: filtering traffic, enabling or disabling links, changing the allocated bandwidth, and redirecting traffic. By combining the SDN actions with NFV and Linux containers we composed multi-stage defences. In Chapter 4 we showed that by only starting analysis functions when they are required and by only redirecting the malicious traffic through the analysis functions we reduce the resources that are required for analysis. Also, any negative impact on the network traffic that may be caused by the analysis function only affects the attack traffic and does not affect the legitimate traffic. The new information provided by the analysis function is used to start a mitigation function that operates on the attack traffic.

Furthermore, the encapsulation of the detection methods and the defences using containers opens up the possibility to share them with other parties and to increase each others defence capabilities.

2. We demonstrated throughout the thesis that a software-defined approach is ideal to set up repeatable experiments which give insights in how attacks affect the network and how to defend against them. Our experimentation environment, VNET, is fully software defined; from a description file of the network we start, monitor, and operate the full software defined overlay network of interconnected virtual machines. We use NFVs such as virtual routers and switches to interconnect the virtual machines. Each VM represents a domain which runs multiple containerised functions for monitoring, for attacking and for defending. Additionally, each domain has an agent that controls the infrastructure and responds to security threats on that domain. Information from all components is collected and visualised in a understandable and user controllable graphical interface for analysis and interaction.

    - RQ3: *Which factors play a role when defending collaboratively, and how do they influence defence efficiency?*

In Chapter 6 we developed three multi-domain defence strategies and we tested their performance under various conditions using *efficiency*. By reflecting on the results, we discovered that analysis times and task dependencies are factors that affect multi-domain defence performance. Also, when analysing the approaches we can see that the network location of the collaborators relative to the attack sources, the placement of countermeasures, and the order of deploying the countermeasures, are factors to consider while developing collaborative defences. The locations relative to the attacker even play a larger role in Chapter 7. By purely using trust to determine at which alliance member to place a countermeasure, the approach can ask help from members that are trustworthy and are not in the path of the attack. By first determining the attack path and subsequently using trust to determine where to place the countermeasures, we can increase the efficiency of such a trust-based approach. We showed that trust and collaboration risk, based on the competence, benevolence, and integrity of the members, are factors that can contribute in selecting the most suitable allies that can help in defending against attacks. Furthermore, we showed that by requesting the most trusted partners for help in defending against the attack has a positive influence on the defence efficiency.

Now that we answered the sub questions we will answer the main research question:

> *Can we create an auto-response system that defends networks against attacks both within a single autonomous domain and within an alliance of domains?*

In this work we proved that it is possible to build auto-response systems that defend networks against attacks. By tightly integrating applications and services and by identifying and monitoring the metrics that indicate the services' performance, we can detect attacks on the network and respond to them using an efficient defence. When forming an alliance of these SARNETs, we can defend using resources provided by allies and defend against attacks closer to the attacker before the attacks become problems that cause service disruption.

8.2 FUTURE WORK

During our explorations and considerations, we identified topics for future research:

8.2.1 *Assessment of risks*

In Chapter 2 we identified *Risk* assessment as part of the SARNET control-loop. This step should assess the risk of applying, or not applying one of the countermeasures. After each attack we evaluate the efficiency of a countermeasure. Efficiency is derived from the impacts on valuable aspects of the system; these impacts can be seen as costs of the system. Since we record the impacts and efficiencies after each attack we eventually form a knowledge base of these historical values. By finding a probability distribution that fits the historical values in the knowledge base, we could statistically determine the probability that a countermeasure has a negative effect on the impact or efficiency. This probability combined with the impact itself gives an indication of the risk of using a countermeasure. The risk that can be used as a predictor and as a decision value in the *risk* step of the SARNET control-loop. Research is required into what probability distributions fit the data in the knowledge base and how assessing these risks affect the performance of the automated defence.

8.2.2 *Analysis and defence function sharing between domains*

In Chapter 4, we show how we are able to use containerised security NFVs in a single domain environment. Besides single-domain use of NFVs, they can also be used in multi-domain environments. Containerised mitigation techniques and detection algorithms can be developed and shared between befriended domains to enhance each others defence capabilities, or can be developed and provided by a trusted third party that specialises in providing such functions. When sharing these functions, we need to ensure they are safe to use and do not contain or use any malicious code that can cause disruptions or information leaks. More research is needed on how to verify the integrity of the containers and the functions within. The risks of running foreign code can be reduced by introducing a third party that verifies the source code and certifies that the code is safe. Using reproducible builds [88], a domain can verify that the software in the packages originates from the verified code. DL4LD [32] and SecConNet [95] are projects in which we research how to provide secure execution environments that promote data and algorithm sharing between organisations in data marketplaces. The outcomes of these projects, and in particular the methods for algorithm and data sharing that ensure confidentiality and integrity, should make it possible to safely execute the NFVs that are shared between the domains.

8.2.3 *Interconnecting SARNETs*

In Chapter 2 we describe three SARNET interconnection patterns. This thesis, however, only focuses on the reactive disjoint SARNET. Besides the reactive disjoint SARNET interconnection pattern, the other interconnection patterns described in Chapter 2 have to be researched. In particular, it has to be studied how the provided patterns allow multi-domain SARNETs to scale and interconnect. Furthermore, it has to be researched how using the interconnection patterns influences the autonomy of the interconnected domains.

Our model for the nested SARNET and intersecting SARNETs involves multiple control-loops. These control-loops are not expected to be completely orthogonal and will influence each other. Hence, it is necessary to study the conditions under which such interacting control loops converge to stable behaviour of the SARNETs. Although many of these issues can be solved by using a nested SARNET pattern in which the parent can take over when its children

show conflicting behaviour, this may affect the autonomy of the children. Therefore, research is needed in how to detect and resolve these conflicts in a distributed manner where each SARNET keeps its autonomy.

8.3 VISION AND OUTLOOK

The Internet where all connected devices can establish end-to-end communication is not sustainable from a security point of view. Insecure and compromised devices from anywhere in the world can communicate with each other and attackers use these systems to launch attacks to disrupt critical services. Compartmentalising the Internet using overlay networks is a suitable approach to reduce the impact of these world scale attacks. We align with the vision of Makkes[66], where he envisions that overlay networks, or virtual internets, can be extended into the users equipment to directly connect the user to the service that is provided by the overlay.

We envision that future Internet Service Providers will provide authentication services and gateways to these overlays. Since all the necessary services to access the overlays will be available in the ISP network, the underlay network will not require global connectivity. Because no global connectivity is required, any malicious activity is contained inside the underlay network of the ISP, who can implement measures to ensure it does not affect the other users. When the right credentials are provided, the user gets permission to access the services' gateway and is able to set-up a connection directly into the overlay of the service.

Providers of these services and applications will have complete autonomy over their overlay network. The service provider can set requirements for the user that wants to access their services and only allow access to the overlay network when users agree to them. Since the service provider has complete control over the overlay network and has control over its users, it can disconnect misbehaving or malicious users from the overlay network to stop attacks on their services.

We see that compartmentalisation is key for sustainable security on the Internet. A logical approach to compartmentalise networks is by using the SARNETs that we described in this thesis. We envision that the future Internet will consist of interconnected SARNETs that securely deliver services to end-users.

Part I

APPENDIX

A

SARNET DEMO USER INTERFACES

This appendix shows how the SARNET user-interface evolved over time. The user-interface was designed by Ben de Graaff with input from other project members. The UI is customly build for SARNET and made for interaction through a multi-touch enabled interface through a web browser. The client side code is built in JavaScript and uses D3.js for drawing graphs.

We demonstrated four versions of the graphical user interface. Each version visualises the parts of the SARNET work that we wanted to emphasize at the time:

- Ciena Vectors Fig. A.1: No automatic response. Mostly visualization. Contained an orchestration interface to build your own network. Allowed the user to implement some basic offensive and defensive actions.
- SC15 Fig. A.2: No automatic response. In this case the user had to enable and disable defenses.
- SC16 Figs. A.3 and A.4: First version to include automatic response. This is done by a single SARNET-agent that has a full view of the network.
- SC17 Figs. A.5 and A.6: Fully includes multi domain features. Each cloud is a separate domain that runs its own sarnet-agent.

For repetitive experiments, we use command line interfaces as a front end. Using this interface, we could access the same options as via the touch-table interfaces and we could execute the same actions as via the touch-table interface to initiate attacks and defenses. The command line interface provided scripting abilities that are used to repeat experiments using different settings. This interface went through the same iterations as the graphical user interfaces and were mainly developed in Python by Ralph Koning.

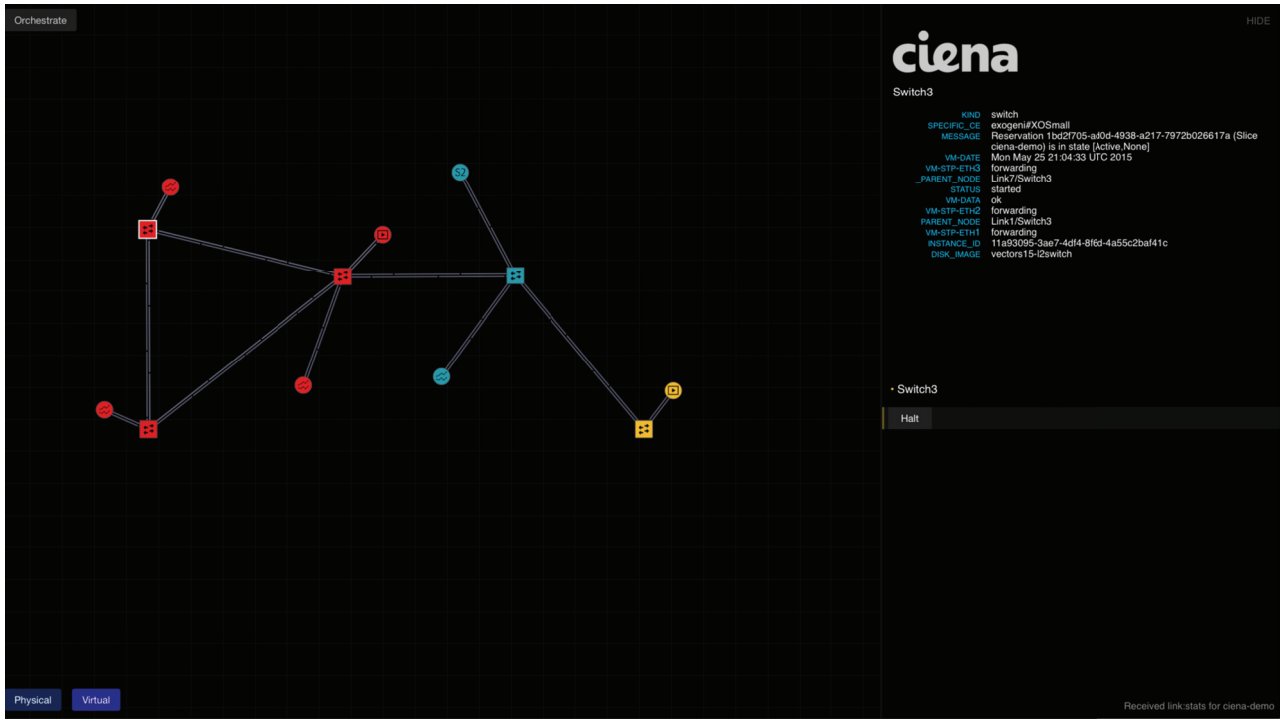


Figure A.1: The user interface during the Ciena VECTORS event in early 2015: We used this interface for visualization and orchestration of the network. The demo included an orchestration-interface (no screenshot available) where one could design their own network. We removed this feature in later versions because it took a considerable amount of time to start the network. The user interface could be used to start and stop data transfers and visualising them. The user was also able to shut down links.

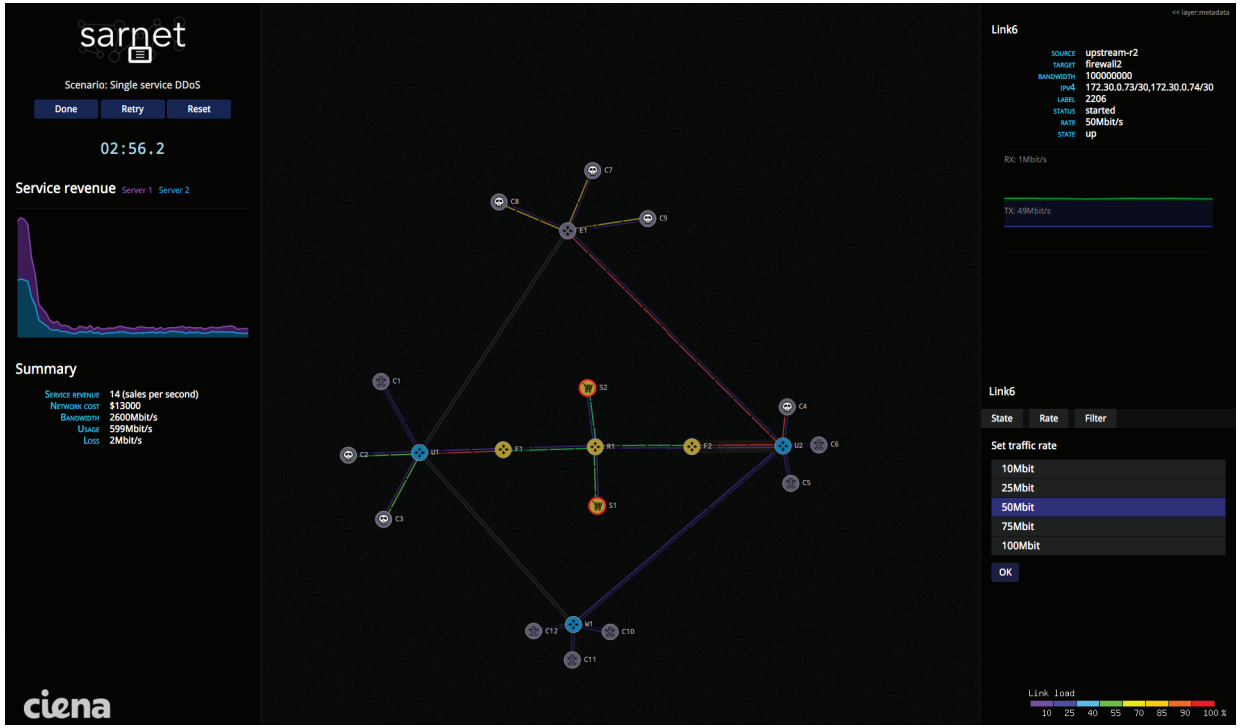


Figure A.2: User interface during Super Computing late 2015: This version exposed the actions of the SDN, shut down link, change bandwidth, apply filter, to the user. The user had a limited amount of time to use the actions in defending against the attacks that were initiated by the system. This version of the demo is thoroughly discussed in Chapter 3.

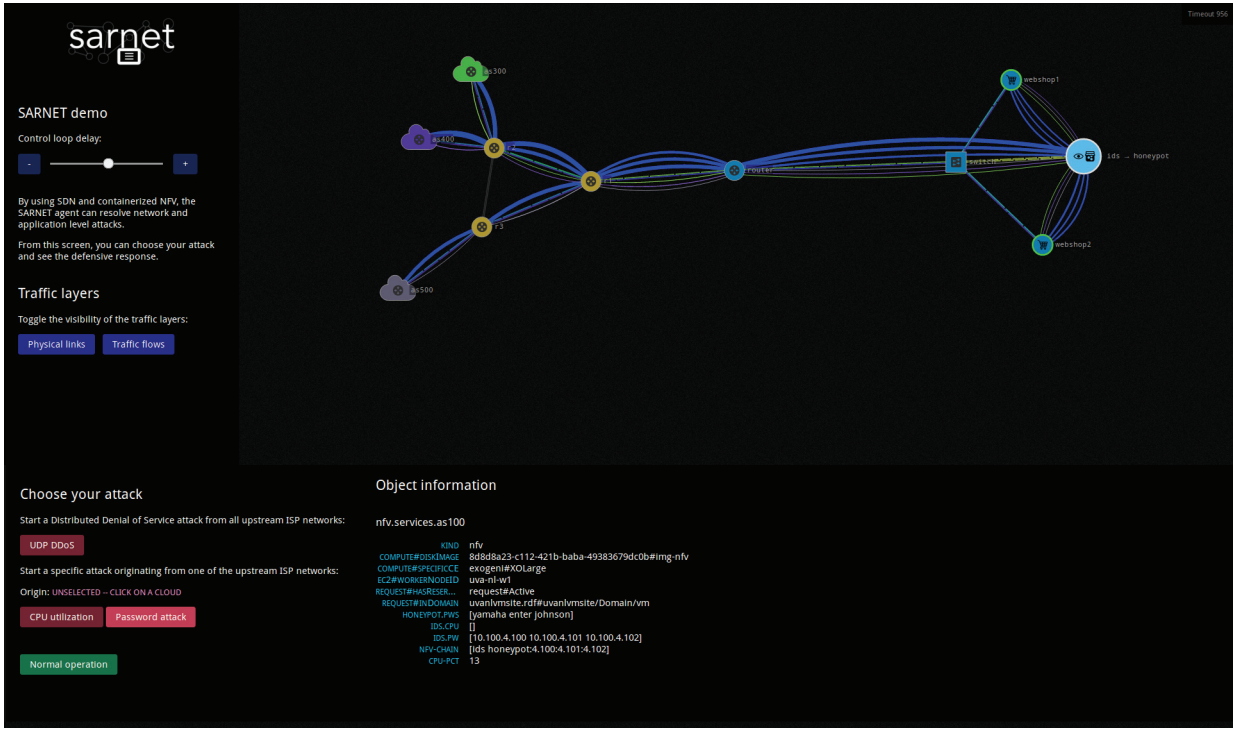


Figure A.3: Touch interface during SC16: The user-interface visualised a single domain and the user could initiate the attack by selecting the type of attack and the attack sources. The demo used the SARNET-agent to defend and we visualised the defence progress. The picture shows a defence that consists of deploying two NFVs: an IDS for analysis, and a honeypot for mitigation and analysis. In this case all the attack traffic is routed through these NFVs using the SDN. This version of the demo is discussed in Chapter 4.

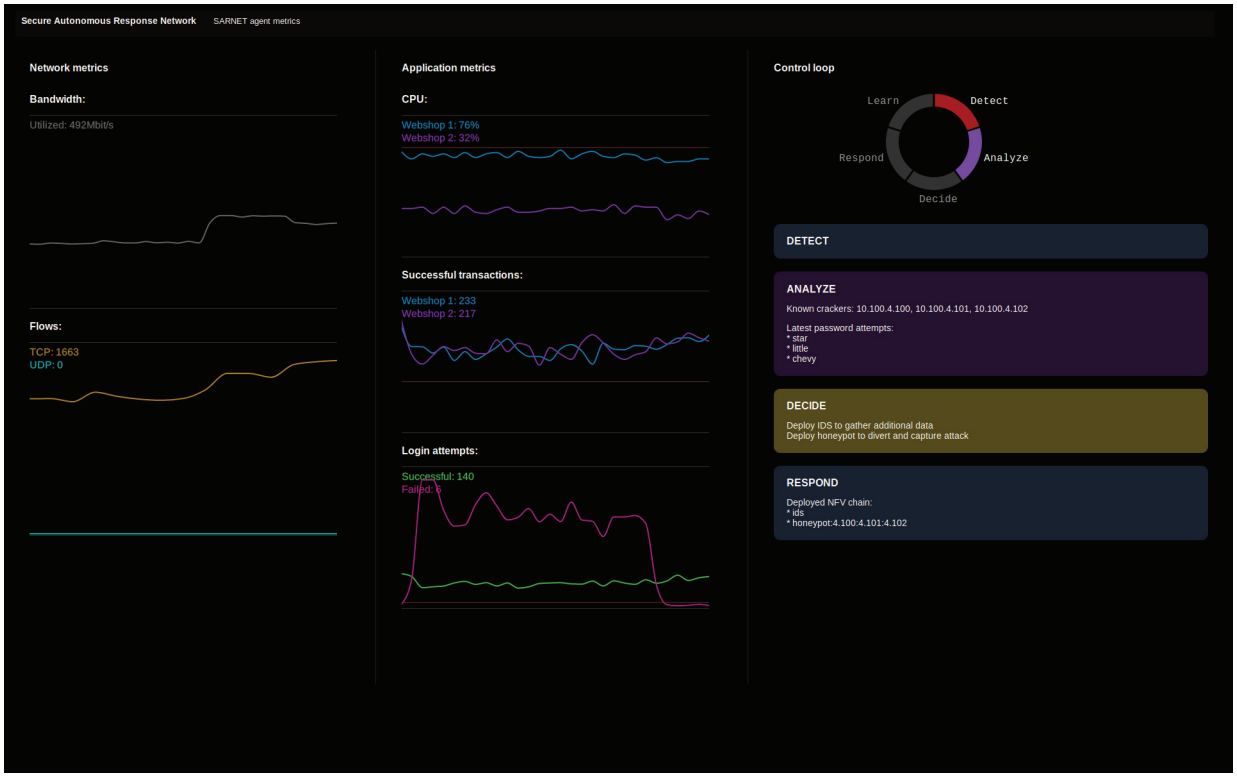


Figure A.4: Metrics interface during SC16: This interface visualises the observables, the security state, and the actions performed by the SARNET-agent and complemented the user interface in Fig. A.3. The interface shows the output of an active password attack. The failed login attempts increased and crossed the threshold. After that, the SARNET-agent starts a defence consisting of a IDS, and a Honeypot, after which the failed login attempts decreased.

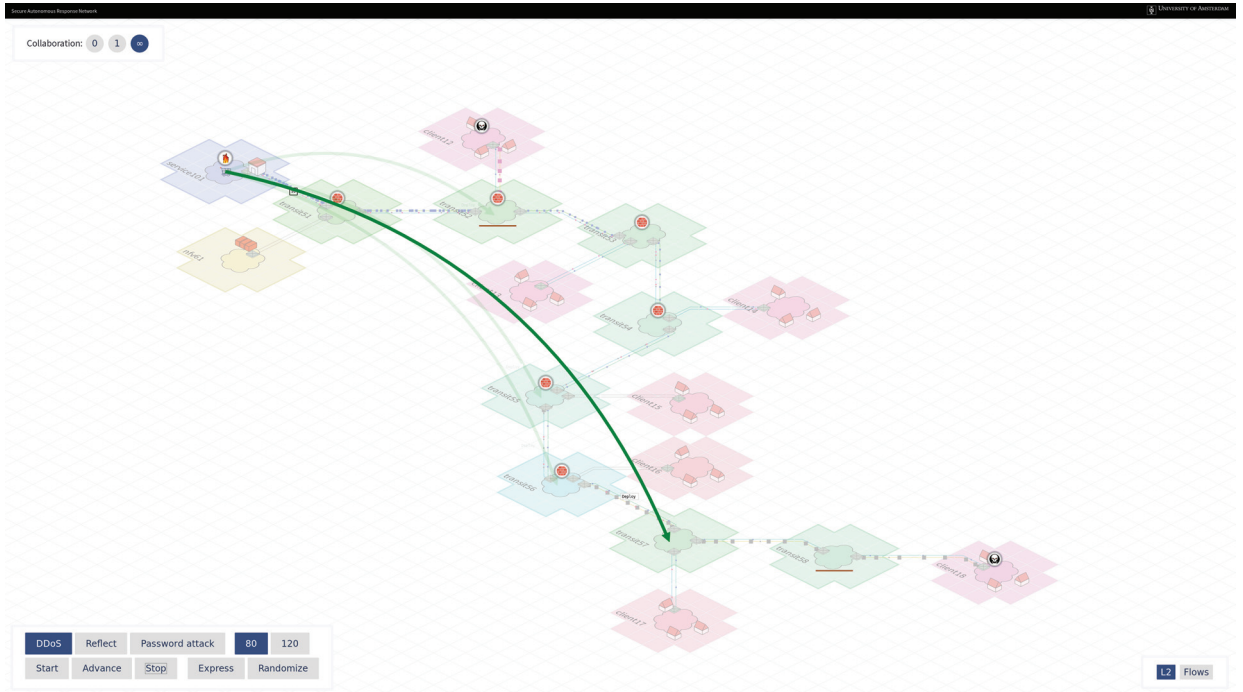


Figure A.5: Touch interface during SC17: The user-interface now visualises multiple domains. Each domain contains the functionality of the previous versions. The user can initiate various type of attacks. This figure, shows the response to a DDoS Attack. Filters are deployed at multiple domains and the victim is continuing to deploy filters at alliance members until a filter is placed at the members close to the attacker. When the intermediate alliance members see that the traffic is filtered upstream, their filters expire.

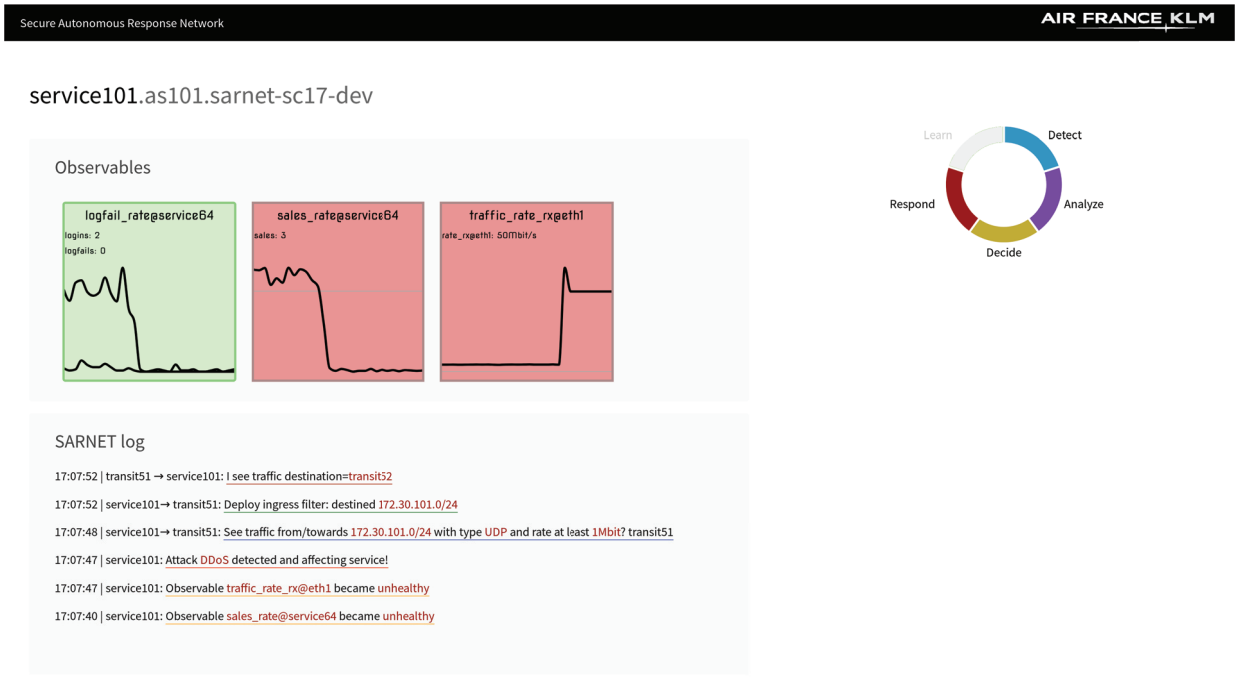


Figure A.6: Metrics interface during SC17 : This interface shows the observables per domain, and a log that indicates what actions are being executed. In this case the interface shows the victim, *service101*, under attack. The sales and the traffic rate observables are violated. The log shows that these observable violations classify as a DDoS, and that multi domain defence tasks are being executed.

```
$ ./vnet-cli -v results
: prepared 2 scenarios, timeout 60. max runtime 900s
: stopping all running attacks
: cleanup: start
: cleanup: done
: learning: start
: learning: ended
: start: pw(service101,['client15', 'client12'],local)
: cleanup: start
: cleanup: done
: runner: start attacks (level=local)
: client: broadcasting domain behaviour parameters
: attack detected at: 3
: runner: victim reports attacked
: timeout at: 63, 60
: runner: timeout occurred
: attack duration: 63s
: runner: waiting a bit
: runner: stopping attacks
: runner: end
```

Figure A.7: Sample output of cli based UI running a scenario of which the results are saved to a file called "results" : The client runs the password attack scenario against *service101* with attackers *client15* and *client12*. In this case the attack has to be resolved locally, within the victim domain. After 60 seconds a timeout occurs stopping the execution of the attack scenario.

BIBLIOGRAPHY

[1] T. Anderson, K. Birman, R. Broberg, M. Caesar, D. Comer, C. Cotton, M. J. Freedman, A. Haeberlen, Z. G. Ives, A. Krishnamurthy, et al. "A Brief Overview of the NEBULA Future Internet Architecture". In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014), pp. 81–86.

[2] G. Androulidakis, V. Chatzigiannakis, and S. Papavassiliou. "Network Anomaly Detection and Classification via Opportunistic Sampling". In: *IEEE network* 23.1 (2009), pp. 6–12.

[3] I. Baldin, J. Chase, Y. Xin, A. Mandal, P. Ruth, C. Castillo, V. Orlikowski, C. Heermann, and J. Mills. "Exogeni: A Multi-Domain Infrastructure-as-a-Service Testbed". In: *The GENI Book*. Springer, 2016, pp. 279–315.

[4] I. Baldine, Y. Xin, A. Mandal, C. H. Renci, J. Chase, V. Marupadi, A. Yumerefendi, and D. Irwin. "Networked Cloud Orchestration: a GENI Perspective". In: *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*. IEEE. 2010, pp. 573–578.

[5] I. Baldine, Y. Xin, A. Mandal, P. Ruth, C. Heerman, and J. Chase. "Exogeni: A Multi-Domain Infrastructure-as-a-Service Testbed". In: *Testbeds and Research Infrastructure. Development of Networks and Communities*. Springer, 2012, pp. 97–113.

[6] S. Barnum. "Standardizing cyber threat intelligence information with the structured threat information expression (stix)". In: *Mitre Corporation* 11 (2012), pp. 1–22.

[7] D. Basin, S. Debois, and T. Hildebrandt. "On purpose and by necessity: compliance under the GDPR". In: *Proceedings of Financial Cryptography and Data Security* 18 (2018).

[8] R. Battula L. "Network Security Function Virtualization (NSFV) towards Cloud computing with NFV Over Openflow infrastructure: Challenges and novel approaches". In: *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2014, pp. 1622–1628. DOI: 10.1109/ICACCI.2014.6968453.

[9] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, et al. "ONOS: towards an open, distributed SDN OS". In: *Proceedings of the third workshop on Hot topics in software defined networking*. ACM. 2014, pp. 1–6.

[10] S. Bhatt, P. K. Manadhata, and L. Zomlot. "The operational role of security information and event management systems". In: *IEEE security & Privacy* 12.5 (2014), pp. 35–41.

[11] G. Boddapati, J. Day, I. Matta, and L. Chitkushev. "Assessing the security of a clean-slate internet architecture". In: *Network Protocols (ICNP), 2012 20th IEEE International Conference on*. IEEE. 2012, pp. 1–6.

[12] E. Boschi, L. Mark, J. Quittek, M. Stiemerling, and P. Aitken. "RFC 5153: IP flow information export (IPFIX) implementation guidelines". In: *IETF, April* (2008).

[13] N. Buraglio and S. Campbell. *ESnet WAN Security Updates*. 2018. URL: https://meetings.internet2.edu/2018-technology-exchange/detail/10005144/. Presentation.

[14] S. Campbell and J. Lee. "Intrusion detection at 100G". In: *State of the Practice Reports*. ACM. 2011, p. 14.

[15] D. Carasso. "Exploring splunk". In: *published by CITO Research, New York, USA, ISBN* (2012), pp. 978–.

[16] M. Chen, Y. Qian, S. Mao, W. Tang, and X. Yang. "Software-Defined Mobile Networks Security". In: *Mobile Networks and Applications* 21.5 (2016), pp. 729–743. ISSN: 1572-8153. DOI: 10.1007/s11036-015-0665-5. URL: http://dx.doi.org/10.1007/s11036-015-0665-5.

[17] Y. Chen, K. Hwang, and W. Ku. "Collaborative Detection of DDoS Attacks over Multiple Network Domains". In: *IEEE Transactions on Parallel & Distributed Systems* 18 (2007), pp. 1649–1662. ISSN: 1045-9219. DOI: 10.1109/TPDS.2007.1111. URL: doi.ieeecomputersociety.org/10.1109/TPDS.2007.1111.

[18] Y. Chen, K. Hwang, and W.-S. Ku. "Distributed change-point detection of DDoS attacks over multiple network domains". In: *Int. Symp. on Collaborative Technologies and Systems*. Citeseer. 2006, pp. 543–550.

[19] J. Claassen, **R. Koning**, and P. Grosso. "Linux containers networking: Performance and scalability of kernel modules". In: *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*. IEEE. IEEE, 2016, pp. 713–717. DOI: 10.1109/NOMS.2016.7502883.

[20] D. D. Clark. "The end-to-end argument and application design: the role of trust". In: *Fed. Comm. LJ* 63 (2010), p. 357.

[21] CloudFlare. *Details of the CloudFlare outage on July 2 2019*. 2019. URL: https://new.blog.cloudflare.com/details-of-the-cloudflare-outage-on-july-2-2019/.

[22] CTA. *Cyber Threat Alliance*. 2019. URL: https://www.cyberthreatalliance.org/.

[23] T. K. Das and B.-S. Teng. "Trust, control, and risk in strategic alliances: An integrated framework". In: *Organization studies* 22.2 (2001), pp. 251–283.

[24] J. Day. *Patterns in network architecture: a return to fundamentals*. Pearson Education, 2007.

[25] H. Debar, M. Dacier, and A. Wespi. "Towards a taxonomy of intrusion-detection systems". In: *Computer Networks* 31.8 (1999), pp. 805–822.

[26] A. Deljoo, **R. Koning**, T. van Engers, L. Gommans, and C. de Laat. "Managing Effective Collaboration in Cyber-security Alliances Using Social Computational Trust". In: *2019 3rd Cyber Security in Networking Conference (CSNet) (CSNet'19) [accepted]*. IEEE. Quito, Ecuador, Oct. 2019.

[27] A. Deljoo, T. van Engers, L. Gommans, and C. de Laat. "Social Computational Trust Model (SCTM): A Framework to Facilitate the Selection of Partners". In: *2018 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS)*. 2018.

[28] A. Deljoo, L. Gommans, C. de Laat, and T. van Engers. "The Service Provider Group Framework". In: *Looking Beyond the Internet: Workshop on Software-defined Infrastructure and Software-defined Exchanges* (2016).

[29] A. Deljoo, T. van Engers, L. Gommans, and C. de Laat. "Social Computational Trust Model (SCTM): A Framework to Facilitate Selection of Partners". In: *2018 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS)*. IEEE. 2018, pp. 45–54.

[30] A. Deljoo, T. van Engers, **R. Koning**, L. Gommans, and C. de Laat. "Towards Trustworthy Information Sharing by Creating Cyber Security Alliances". In: *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE. Aug. 2018, pp. 1506–1510. DOI: 10.1109/TrustCom/BigDataSE.2018.00213.

[31] F. Dijkstra. "Framework for Path Finding in Multi-Layer Transport Networks". PhD thesis. University of Amsterdam, 2009.

[32] *DL4LD: Data Logistics for Logistics Data*. URL: https://www.dl4ld.net/.

[33] T. Dübendorfer, A. Wagner, and B. Plattner. "An economic damage model for large-scale internet attacks". In: *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2004. WET ICE 2004. 13th IEEE International Workshops on*. IEEE. 2004, pp. 223–228.

[34] P. Ferguson. *Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing*. 2000.

[35] M. Ghijsen, J. Van der Ham, P. Grosso, and C. De Laat. "Towards an infrastructure description language for modeling computing infrastructures". In: *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*. IEEE. 2012, pp. 207–214.

[36] G. G. Granadillo, M. Belhaouane, H. Debar, and G. Jacob. "RORI-based countermeasure selection using the OrBAC formalism". In: *International journal of information security* 13.1 (2014), pp. 63–79.

[37] C. Guok, D. Robertson, M. Thompson, J. Lee, B. Tierney, and W. Johnston. "Intra and interdomain circuit provisioning using the oscars reservation system". In: *2006 3rd International Conference on Broadband Communications, Networks and Systems*. IEEE. 2006, pp. 1–8.

[38] B. Han, V. V. Gopalakrishnan, L. Ji, and S. Lee. "Network function virtualization: Challenges and opportunities for innovations". In: *IEEE Communications Magazine* 53.2 (2015), pp. 90–97. ISSN: 0163-6804. DOI: 10.1109/MCOM.2015.7045396.

[39] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal. "NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC)". In: *IEEE Network* 28.6 (2014), pp. 18–26. ISSN: 0890-8044. DOI: 10.1109/MNET.2014.6963800.

[40] J. Hill, M. Aloserij, and P. Grosso. "Tracking network flows with P4". In: *2018 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS)*. IEEE. 2018, pp. 23–32.

[41] Internet Society. *ISOC Global Internet Report 2017*. URL: https://www.internetsociety.org/globalinternetreport/2017/.

[42] J. H. Jafarian, E. Al-Shaer, and Q. Duan. "Openflow random host mutation: transparent moving target defense using software defined networking". In: *Proceedings of the first workshop on Hot topics in software defined networks*. ACM. 2012, pp. 127–132.

[43] P. Jakma and D. Lamparter. "Introduction to the quagga routing suite". In: *IEEE Network* 28.2 (2014), pp. 42–48.

[44] M. Jonker, A. King, J. Krupp, C. Rossow, A. Sperotto, and A. Dainotti. "Millions of targets under attack: a macroscopic characterization of the DoS ecosystem". In: *Proceedings of the 2017 Internet Measurement Conference*. ACM. 2017, pp. 100–113.

[45] R. Kesavamoorthy and K. R. Soundar. "Swarm intelligence based autonomous DDoS attack detection and defense using multi agent system". In: *Cluster Computing* (2018), pp. 1–8.

[46] N. Kheir, N. Cuppens-Boulahia, F. Cuppens, and H. Debar. "A service dependency model for cost-sensitive intrusion response". In: *Computer Security–ESORICS 2010* (2010), pp. 626–642.

[47] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou. "Feature-based comparison and selection of Software Defined Networking (SDN) controllers". In: *Computer Applications and Information Systems (WCCAIS), 2014 World Congress on*. IEEE. 2014, pp. 1–7.

[48] G. Killcrece, K.-P. Kossakowski, R. Ruefle, and M. Zajicek. *State of the practice of computer security incident response teams (CSIRTs)*. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2003.

[49] **R. Koning**, B. de Graaff, G. Polevoy, R. Meijer, C. de Laat, and P. Grosso. "Measuring the Efficiency of SDN Mitigations Against Attacks on Computer Infrastructures". In: *Future Generation Computer Systems* 91 (2019), pp. 144–156. DOI: 10.1016/j.future.2018.08.011.

[50] **R. Koning**, A. Deljoo, L. Meijer, C. de Laat, and P. Grosso. "Trust-based Collaborative Defences in Multi Network Alliances". In: *2019 3rd Cyber Security in Networking Conference (CSNet) (CSNet'19) [accepted]*. IEEE. Quito, Ecuador, Oct. 2019.

[51] **R. Koning**, G. Polevoy, L. Meijer, C. de Laat, and P. Grosso. "Approaches for Collaborative Security Defences in Multi Network Environments". In: *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. June 2019, pp. 113–123. DOI: 10.1109/CSCloud/EdgeCom.2019.000-9.

[52] **R. Koning**, N. Buraglio, C. de Laat, and P. Grosso. "CoreFlow: Enriching Bro security events using network traffic monitoring data". In: *Future Generation Computer Systems* 79 (2018), pp. 235–242. DOI: 10.1016/j.future.2017.04.017.

[53] **R. Koning**, B. de Graaff, C. de Laat, R. Meijer, and P. Grosso. "Interactive analysis of SDN-driven defence against Distributed Denial of Service attacks". In: *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. IEEE. June 2016, pp. 483–488. DOI: 10.1109/NETSOFT.2016.7502489.

[54] **R. Koning**, B. de Graaff, R. Meijer, C. de Laat, and P. Grosso. "Measuring the Effectiveness of SDN Mitigations against Cyber Attacks". In: *2017 IEEE Conference on Network Softwarization (NetSoft)*. IEEE. July 2017, pp. 1–6. DOI: 10.1109/NETSOFT.2017.8004231.

[55] **R. Koning**, P. Grosso, and C. de Laat. "Using ontologies for resource description in the CineGrid Exchange". In: *Future Generation Computer Systems* 27.7 (2011), pp. 960–965. DOI: 10.1016/j.future.2010.11.027.

[56] **R. Koning** et al. "Enabling E-Science Applications with Dynamic Optical Networks: Secure Autonomous Response Networks". In: *Optical Fiber Communication Conference*. Optical Society of America. 2017, Tu3E–1. DOI: 10.1364/OFC.2017.Tu3E.1.

[57] T. R. Koscik and D. Tranel. "The human amygdala is necessary for developing and expressing normal interpersonal trust". In: *Neuropsychologia* 49.4 (2011), pp. 602–611.

[58] D. Kreutz, F. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. "Software-defined networking: A comprehensive survey". In: *arXiv preprint arXiv:1406.0440* (2014).

[59] A. Lara and B. Ramamurthy. "OpenSec: Policy-based security using software-defined networking". In: *IEEE transactions on network and service management* 13.1 (2016), pp. 30–42.

[60] F.-Y. Lee, S. Shieh, J.-T. Shieh, and S.-H. Wang. "A Source-End Defense System Against DDoS Attacks". In: *Computer Security in the 21st Century*. Ed. by D. T. Lee, S. P. Shieh, and J. D. Tygar. Boston, MA: Springer US, 2005, pp. 147–168. ISBN: 978-0-387-24006-0. DOI: 10.1007/0-387-24006-3_10. URL: https://doi.org/10.1007/0-387-24006-3_10.

[61] V. Lenders, A. Tanner, and A. Blarer. "Gaining an Edge in Cyberspace with Advanced Situational Awareness". In: *IEEE Security & Privacy* 2 (2015), pp. 65–74.

[62] F. Lesueur, A. Rezmerita, T. Herault, S. Peyronnet, and S. Tixeuil. "SAFE-OS: A secure and usable desktop operating system". In: *Risks and Security of Internet and Systems (CRiSIS), 2010 Fifth International Conference on*. IEEE. 2010, pp. 1–7.

[63] D. Z. Levin, R. Cross, L. C. Abrams, and E. L. Lesser. "Trust and knowledge sharing: A critical combination". In: *IBM Institute for Knowledge-Based Organizations* 19 (2002).

[64] J. MacAuley. *ESnet6: Orchestrating the Programmable Network*. 2018. URL: https://tnc18.geant.org/core/presentation/148. Presentation.

[65] M. V. Mahoney. "A machine learning approach to detecting attacks by identifying anomalies in network traffic". PhD thesis. Florida Institute of Technology, 2003.

[66] M. X. Makkes. "Virtual Internets". PhD thesis. University of Amsterdam, 2018.

[67] S. Mannhart, B. Rodrigues, E. Scheid, S. S. Kanhere, and B. Stiller. "Toward Mitigation-as-a-Service in Cooperative Network Defenses". In: *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE. 2018, pp. 362–367.

[68] S. Mansfield-Devine. "Under the radar". In: *Network Security* 2015.12 (2015), pp. 14–18.

[69] Matthew Prince – CloudFlare. *The Relative Cost of Bandwidth Around the World*. 2016. URL: https://blog.cloudflare.com/the-relative-cost-of-bandwidth-around-the-world/.

[70] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. "OpenFlow: enabling innovation in campus networks". In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 69–74.

[71] G. Meng, Y. Liu, J. Zhang, A. Pokluda, and R. Boutaba. "Collaborative Security: A Survey and Taxonomy". In: *ACM Comput. Surv.* 48.1 (2015), 1:1–1:42. ISSN: 0360-0300. DOI: 10.1145/2785733. URL: http://doi.acm.org/10.1145/2785733.

[72] J. Mirkovic, G. Prier, and P. Reiher. "Attacking DDoS at the source". In: *10th IEEE International Conference on Network Protocols, 2002. Proceedings.* 2002, pp. 312–321. DOI: 10.1109/ICNP.2002.1181418.

[73] C. Morrow and R. Dobbins. "DOTS (DDoS Open Threat Signaling) Working Group Operational Requirements". In: *IETF* 93 (2015).

[74] W. Nejdl, D. Olmedilla, and M. Winslett. "PeerTrust: Automated Trust Negotiation for Peers on the Semantic Web". In: *Secure Data Management*. Ed. by W. Jonker and M. Petković. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 118–132. ISBN: 978-3-540-30073-1.

[75] NetFlow, Cisco IOS. *Introduction to Cisco IOS NetFlow A Technical Overview*. accessed on 2016/09/15. 2012.

[76] Network Weathermap. 2016. URL: http://network-weathermap.com/.

[77] OASIS. *Open Command and Control (OpenC2)*. 2019. URL: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=openc2.

[78] Packet Design. *Route Explorer*. accessed on 2016/09/15. 2016.

[79] V. Paxson. "Bro: a system for detecting network intruders in real-time". In: *Computer networks* 31.23 (1999), pp. 2435–2463.

[80] T. Peng, C. Leckie, and K. Ramamohanarao. "Survey of Network-based Defense Mechanisms Countering the DoS and DDoS Problems". In: *ACM Comput. Surv.* 39.1 (2007). ISSN: 0360-0300. DOI: 10.1145/1216370.1216373. URL: http://doi.acm.org/10.1145/1216370.1216373.

[81] G. Polevoy. *Defence Efficiency*. 2019. arXiv: 1904.07141.

[82] E. Pouyoul. *ENOS: A Network Operating System for ESnet Testbed*.

[83] P. Quinn and T. Nadeau. "Service function chaining problem statement". In: *IEEF RFC 7498* (2015).

[84] L. Rainie and J. Anderson. *The Fate of Online Trust in the Next Decade*. URL: http://www.pewinternet.org/2017/08/10/the-fate-of-online-trust-in-the-next-decade.

[85] S. Ranjan, J. Robinson, and F. Chen. *Machine learning based botnet detection using real-time connectivity graph based traffic features*. US Patent 8,762,298. 2014. URL: https://www.google.com/patents/US8762298.

[86] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker. "Modular sdn programming with pyretic". In: *Technical Reprot of USENIX* (2013).

[87] Y. Rekhter, T. Li, and S. Hares. *A border gateway protocol 4 (BGP-4)*. Tech. rep. ietf, 2005.

[88] Reproducible builds. *Reproducible builds: a set of software development practices that create an independently-verifiable path from source to binary*. 2019. URL: https://reproducible-builds.org/.

[89] J. Rutkowska and R. Wojtczuk. "Qubes OS architecture". In: *Invisible Things Lab Tech Rep* (2010), p. 54.

[90] T. Ryutov, L. Zhou, C. Neuman, N. Foukia, T. Leithead, and K. E. Seamons. "Adaptive trust negotiation and access control for grids". In: *The 6th IEEE/ACM International Workshop on Grid Computing, 2005*. 2005. DOI: 10.1109/GRID.2005.1542724.

[91] S. Scott-Hayward, G. O'Callaghan, and S. Sezer. "SDN Security: A Survey". In: *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*. 2013, pp. 1–7. DOI: 10.1109/SDN4FNS.2013.6702553.

[92] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao. "Are we ready for SDN? Implementation challenges for software-defined networks". In: *Communications Magazine, IEEE* 51.7 (2013), pp. 36–43. ISSN: 0163-6804. DOI: 10.1109/MCOM.2013.6553676.

[93] A. M. Shabut, K. P. Dahal, S. K. Bista, and I. U. Awan. "Recommendation based trust model with an effective defence scheme for MANETs". In: *IEEE Transactions on mobile computing* 14.10 (2014), pp. 2101–2115.

[94] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson. "FRESCO: Modular Composable Security Services for Software-Defined Networks." In: *NDSS*. 2013.

[95] *Smart, secure container networks for trusted Big Data Sharing*. Netherlands eScience center. URL: https://www.esciencecenter.nl/project/secconnet.

[96] R. Sommer and V. Paxson. "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection". In: *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE. 2010, pp. 305–316. DOI: 10.1109/SP.2010.25.

[97] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. "An overview of IP flow-based intrusion detection". In: *IEEE communications surveys & tutorials* 12.3 (2010), pp. 343–356.

[98] W. Stallings. *Handbook of computer-communications standards; Vol. 1: the open systems interconnection (OSI) model and OSI-related standards*. Macmillan Publishing Co., Inc., 1987.

[99] J. Steinberger, B. Kuhnert, C. Dietz, L. Ball, A. Sperotto, H. Baier, A. Pras, and G. Dreo. "DDoS Defense using MTD and SDN". In: *16th IEEE/IFIP Network Operations and Management Symposium 2018: Cognitive Management in a Cyber World*. 2018.

[100] J. Steinberger, B. Kuhnert, A. Sperotto, H. Baier, and A. Pras. "Collaborative DDoS defense using flow-based security event information." In: *NOMS*. Vol. 2016. 2016, pp. 516–522.

[101] J. Strassner, N. Agoulmine, and E. Lehtihet. "Focale: A novel autonomic networking architecture". In: *Latin American Autonomic Computing Symposium (LAACS), 2006*. 2006.

[102] R. J. Strijkers. "Internet Factories". PhD thesis. University of Amsterdam, 2013.

[103] R. Strijkers, M. X. Makkes, C. de Laat, and R. Meijer. "Internet factories: Creating application-specific networks on-demand". In: *Computer Networks* 68 (2014). Communications and Networking in the Cloud, pp. 187–198. ISSN: 1389-1286. DOI: http://dx.doi.org/10.1016/j.comnet.2014.01.009. URL: http://www.sciencedirect.com/science/article/pii/S1389128614000395.

[104] N. Sultana, N. Chilamkurti, W. Peng, and R. Alhadad. "Survey on SDN based network intrusion detection system using machine learning approaches". In: *Peer-to-Peer Networking and Applications* 12.2 (2019), pp. 493–501.

[105] S. Suresh and N. S. Ram. "Feasible DDoS attack source traceback scheme by deterministic multiple packet marking mechanism". In: *The Journal of Supercomputing* (2018), pp. 1–15.

[106] S. Tilkov. "The Modern Cloud-Based Platform". In: *IEEE Software* 2 (2015), pp. 116–116.

[107] J. Touch, I. Baldine, R. Dutta, G. G. Finn, B. Ford, S. Jordan, D. Massey, A. Matta, C. Papadopoulos, P. Reiher, et al. "A dynamic recursive unified internet design (DRUID)". In: *Computer Networks* 55.4 (2011), pp. 919–935.

[108] J. Van Der Ham, F. Dijkstra, P. Grosso, R. Van Der Pol, A. Toonk, and C. De Laat. "A distributed topology information system for optical networks based on the semantic web". In: *Optical Switching and Networking* 5.2 (2008), pp. 85–93.

[109] L. M. Vaquero, L. Rodero-Merino, and R. Buyya. "Dynamically scaling applications in the cloud". In: *ACM SIGCOMM Computer Communication Review* 41.1 (2011), pp. 45–52.

[110] T. Vissers, T. Van Goethem, W. Joosen, and N. Nikiforakis. "Maneuvering around clouds: Bypassing cloud-based security providers". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2015, pp. 1530–1541.

[111] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou. "DDoS attack protection in the era of cloud computing and software-defined networking". In: *Computer Networks* 81 (2015), pp. 308–319.

[112] World Economic Forum. *Global Risks Report 2018*. URL: https://www.weforum.org/reports/the-global-risks-report-2018.

[113] D. Xu and P. Ning. "Alert correlation through triggering events and common resources". In: *Computer Security Applications Conference, 2004. 20th Annual*. IEEE. 2004, pp. 360–369.

[114] Q. Yan, F. R. Yu, Q. Gong, and J. Li. "Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges". In: *IEEE Communications Surveys Tutorials* 18.1 (2016), pp. 602–622. ISSN: 1553-877X. DOI: 10.1109/COMST.2015.2487361.

[115] Y. Yao, Q. Cao, J. Chase, P. Ruth, I. Baldin, Y. Xin, and A. Mandal. "Slice-based network transit service: Inter-domain l2 networking on exogeni". In: *Computer Communications Workshops (INFOCOM WKSHPS), 2017 IEEE Conference on*. IEEE. 2017, pp. 736–741.

[116] S. T. Zargar, J. Joshi, and D. Tipper. "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks". In: *IEEE Communications Surveys Tutorials* 15.4 (2013), pp. 2046–2069. ISSN: 1553-877X. DOI: 10.1109/SURV.2013.031413.00127.

[117] Z. Zhao, Y. Guo, and W. Liu. "The Design and Research for Network Address Space Randomization in OpenFlow Network". In: *Journal of Computer and Communications* 3.05 (2015), p. 203.

[118] L. Zhu, X. Tang, M. Shen, X. Du, and M. Guizani. "Privacy-Preserving DDoS Attack Detection Using Cross-Domain Traffic in Software Defined Networks". In: *IEEE Journal on Selected Areas in Communications* 36.3 (2018), pp. 628–643.

ACRONYMS

AI	Artificial Intelligence
API	Application Programming Interface
AS	Autonomous System
BGP	Border Gateway Protocol
CDN	Content Delivery Network
CEASE	Correlation Evaluation and Security Enforcement
CPU	Central Processing Unit
DDoS	Distributed Denial of Service
DIF	Distributed IPC Facility
DPI	Deep Packet Inspection
DSL	Domain Specific Language
DoS	Denial of Service
ESnet	Energy Sciences network
FDD	FulfilleD with Delay
FD	FulfilleD
GENI	Global Environment for Network Innovations
I/O	Input/Output
ICT	Information and Communications Technology
IDS	Intrusion Detection System
ID	IDentifier
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
IO	Input Output
IPC	Inter-Process Communication
IPFIX	IP Flow Information eXport
IP	Internet Protocol address
ISP	Internet Service Provider
IaaS	Infrastructure as a Service

JSON	JavaScript Object Notation
LAN	Local Area Network
MPLS	Multi Protocol Label Switching
MQTT	Message Queuing Telemetry Transport
NFS	Network File System
NFV	Network Function Virtualization
NREN	National Research and Education Network
OSCARS	On-Demand Secure Circuits and Advanced Reservation System
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
OS	Operating System
QoS	Quality of Service
RAM	Random Access Memory
REST	Representational State Transfer
RINA	Recursive InterNetworking Architecture
RIP	Routing Information Protocol
RNA	Recursive InterNetwork Architecture
SARNET	Secure Autonomous Response NETworks
SCTM	Social Computational Trust Model
SC	SuperComputing
SDI	Software Defined Infrastructure
SDN	Software Defined Networking
SFC	Service Function Chaining
SIEM	Security Information and Event Management
SNE	System and Network Engineering research group of UvA
SP	Service Provider
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UI	User Interface

UPVN	User Programmable Virtual Network
URI	Uniform Resource Identifier
URN	Uniform Resource Name
UUID	Universally Unique IDentifier
UvA	Universiteit van Amsterdam
VLAN	Virtual LAN
VM	Virtual Machine
VNET	Virtual sarNET
VNF	Virtual Network Function
VPN	Virtual Private Network
V	Violation (not fulfilled)

PUBLICATIONS

[1] J. Claassen, **R. Koning**, and P. Grosso. "Linux containers networking: Performance and scalability of kernel modules". In: *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*. IEEE. IEEE, 2016, pp. 713–717. DOI: 10.1109/NOMS.2016.7502883.

[2] A. Deljoo, **R. Koning**, T. van Engers, L. Gommans, and C. de Laat. "Managing Effective Collaboration in Cyber-security Alliances Using Social Computational Trust". In: *2019 3rd Cyber Security in Networking Conference (CSNet) (CSNet'19) [accepted]*. IEEE. Quito, Ecuador, Oct. 2019.

[3] A. Deljoo, T. van Engers, **R. Koning**, L. Gommans, and C. de Laat. "Towards Trustworthy Information Sharing by Creating Cyber Security Alliances". In: *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE. Aug. 2018, pp. 1506–1510. DOI: 10.1109/TrustCom/BigDataSE.2018.00213.

[4] **R. Koning**, B. de Graaff, G. Polevoy, R. Meijer, C. de Laat, and P. Grosso. "Measuring the Efficiency of SDN Mitigations Against Attacks on Computer Infrastructures". In: *Innovating the Network for Data Intensive Science (INDIS) workshop at Super Computing 2017, Denver (CO)* (2017).

[5] **R. Koning**, B. de Graaff, G. Polevoy, R. Meijer, C. de Laat, and P. Grosso. "Measuring the Efficiency of SDN Mitigations Against Attacks on Computer Infrastructures". In: *Future Generation Computer Systems* 91 (2019), pp. 144–156. DOI: 10.1016/j.future.2018.08.011.

[6] **R. Koning**, A. Deljoo, L. Meijer, C. de Laat, and P. Grosso. "Trust-based Collaborative Defences in Multi Network Alliances". In: *2019 3rd Cyber Security in Networking Conference (CSNet) (CSNet'19) [accepted]*. IEEE. Quito, Ecuador, Oct. 2019.

[7] **R. Koning**, G. Polevoy, L. Meijer, C. de Laat, and P. Grosso. "Approaches for Collaborative Security Defences in Multi Network Environments". In: *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. June 2019, pp. 113–123. DOI: 10.1109/CSCloud/EdgeCom.2019.000-9.

[8] **R. Koning**, N. Buraglio, C. de Laat, and P. Grosso. "CoreFlow: Enriching Bro security events using network traffic monitoring data". In: *Innovating the Network for Data Intensive Science (INDIS) workshop at Super Computing 2016, Salt Lake City (UT)* (2016).

[9] **R. Koning**, N. Buraglio, C. de Laat, and P. Grosso. "CoreFlow: Enriching Bro security events using network traffic monitoring data". In: *Future Generation Computer Systems* 79 (2018), pp. 235–242. DOI: 10.1016/j.future.2017.04.017.

[10] **R. Koning**, B. de Graaff, C. de Laat, R. Meijer, and P. Grosso. "Interactive analysis of SDN-driven defence against Distributed Denial of Service attacks". In: *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. IEEE. June 2016, pp. 483–488. DOI: 10.1109/NETSOFT.2016.7502489.

[11] **R. Koning**, B. de Graaff, R. Meijer, C. de Laat, and P. Grosso. "Measuring the Effectiveness of SDN Mitigations against Cyber Attacks". In: *2017 IEEE Conference on Network Softwarization (NetSoft)*. IEEE. July 2017, pp. 1–6. DOI: 10.1109/NETSOFT.2017.8004231.

[12] **R. Koning**, P. Grosso, and C. de Laat. "Using ontologies for resource description in the CineGrid Exchange". In: *Future Generation Computer Systems* 27.7 (2011), pp. 960–965. DOI: 10.1016/j.future.2010.11.027.

[13] **R. Koning** et al. "Enabling E-Science Applications with Dynamic Optical Networks: Secure Autonomous Response Networks". In: *Optical Fiber Communication Conference*. Optical Society of America. 2017, Tu3E–1. DOI: 10.1364/OFC.2017.Tu3E.1.

TEACHING AND SUPERVISION

Teaching experience

- Master Security and Network Engineering *Advanced Networking* 2018 Lab teacher
- Master System and Network Engineering *Advanced Networking* 2017 Lab teacher
- Master System and Network Engineering *Essential Skills* 2016 Lab teacher

Thesis supervision

- Tom Carpay, *Network Based Intrusion Detection via Analysis of HTTPS traffic* Bachelor Artificial Intelligence 2018
- Erik Kooistra, *Hardening Virtual Environments Against Cache Based Side-channel Attacks* Bachelor Computer Science 2018
- Bernardus Jansen, *Security By Default - A Comparative Security Evaluation of Default Configurations* SNE master 2018 Co-supervisor Tobias Fiebig (TU Delft)
- Jeffrey Panneman, *eBPF filter acceleration for arbitrary packet matching in the Linux kernel* Software Engineering master 2017 Co-supervisor Piotr Zuraniewski (TNO)
- Sebastiaan Hoekstra, *Defining the optimal route for outbound SMS Thesis supervision* Data Science master 2017
- Leandro Velasco and Nick de Bruijn, *Reliable Library Identification Using VMI Techniques* SNE master 2017.
- Koen Veelenturf, *Taking a closer look at IRATI* SNE master 2016 Co-supervisor Marijke Kaat (SURFnet)
- Jeroen van Leur and Jeroen Klomp, *Recursive InterNetwork Architecture; An Assessment of the IRATI Implementation* SNE master 2016 Co-supervisor Marijke Kaat (SURFnet)
- Andreas Karakannas and Anastasios Poulidis, *ExoGENI: Evaluating the Network Performance of ExoGENI Cloud Computing System* SNE master 2015

SOURCE CODE AND DATASETS

The source code for the SARNET project is published at https://bitbucket.org/uva-sne/projects/SAR/. The source code in these repositories is developed by Ben de Graaff and Ralph Koning. Here, we hilghight the main components:

VNET and the user interface code:

- SC15 version of VNET+UI: https://bitbucket.org/uva-sne/vnet/src/sc15/
- SC16 version of VNET+UI: https://bitbucket.org/uva-sne/vnet/src/sc16-updates/
- SC17 version of VNET+UI: https://bitbucket.org/uva-sne/vnet/src/master/
- 2018/2019 version of VNET+UI with updates for trust: https://bitbucket.org/uva-sne/vnet/src/rk-2018/
- 2018/2019 CLI: https://bitbucket.org/uva-sne/vnet-ui-cli/

SARNET-agent code:

- SC16 version of VNET used for Chapter 4: https://bitbucket.org/uva-sne/sarnet-agent/src/ralph-v1-2016
- SC17 version of SARNET-agent: https://bitbucket.org/uva-sne/sarnet-agent/src/multi-domain
- SC18 version of SARNET-agent used for Chapters 6 and 7: https://bitbucket.org/uva-sne/sarnet-agent/src/ralph-md

Domain components and other tools:

- Tools for starting slices: https://bitbucket.org/uva-sne/xo-tools/
- Scripts building VM images and containers: https://bitbucket.org/uva-sne/vnet-buildscripts/
- Infrastructure controller for a domain: https://bitbucket.org/uva-sne/host-netctl/
- Monitoring component of a domain https://bitbucket.org/uva-sne/vnetmon/

The source code for CoreFlow is published in a private repository listed below. Please contact ESnet for access:

- https://github.com/esnet/CoreFlow/

Published datasets:

- https://bitbucket.org/uva-sne/phdthesis-rkoning-data/

Please note that these links are non-permanent and may change or disappear anytime.

PROJECT ACKNOWLEDGMENTS

We would like to thank the following project partners and other organizations for making this thesis a reality:

- The Dutch Science Foundation NWO for funding the project (grant no: CYBSEC.14.003 / 618.001.016)
- COMMIT for funding part of the project (WP20.11).
- CIENA for providing financial contributions, equipment, discussions and hosting our work at their booth at various conferences.
- TNO for providing input and funding and project management.
- KLM for the interesting use-cases and input.
- LBNL and ESnet for hosting me during my internship and covering my expenses when abroad.
- RENCI and ExoGENI team for their support and prompt response to questions about their platform.
- SURF and NetherLight for providing infrastructure and connectivity and in particular Gerben and Migiel for their support.

ACKNOWLEDGMENTS

Foremost, I would like to thank my partner Rian for her love, her contributions, and for putting up with me during the ups and downs of the PhD process. Paola, my supervisor, for always being able to schedule time for me, her input, and her help in finding alternative approaches when stuck as well as the laughs we had during more casual conversations. Cees and Lydia, thank you for accepting me as your PhD student and thank you for the valuable input and guidance during my research and the writing. Tom, Rob, Leon, Piotr, Pieter, Aiko and Ilia thank you for reading my work and for taking part of my graduation committee.

Ameneh, Ben, Stojan and Gleb collaborated closely with me on the SARNET project; I learned a lot from our interactions and thank you for your input and contributing to my work. Also thanks to the persons that we interacted with during the various SARNET meetings: Frank, Esther, Tako and Arjan. The team from Ciena which also contributed to the project but with whom I worked with for many years before, Marc, Rodney, Gaurav-Deep, thank you for your valuable input and your enthusiasm to our ideas.

I would like to thank all my colleagues in SNE lab that contributed to wonderful PhD experience. Especially, "The proper suspects", you know why, you know what, and you know where ;-). The people I interacted with during lunch, coffee breaks, and other group events: Ana, Ana, Andy, Benjamin, Dolly, Daan, Giovanni, Giulio, Edwin, Grace, Huan, Hugo, Jamila, Jaqueline, Julius, Jun, Lu, Lu-Chi, Marijke, Mary, Merijn, Milen, Misha, Mostafa, Onno, Pieter, Reggie, Saba, Sara, Sander, Simon, Spiros, Tim, Toto, Uraz, Xiaofeng, Xin, Yang. The colleagues that were there when I joined SNE group back in in 2007 (and are still there): Adam, Arie, Yuri and Zhiming. The OS3 team: Karst, Jaap, Arno, Niels, and Peter. My former colleagues that contributed to a great work experience of which some became valuable friends: Canh, Carol, Cosmin, Damien, Daniel, Fahimeh, Fatih, Fred, Freek, Gerben, Hans, Hao, Jeroen, Karel, Koot, Marc, Mattijs, Mihai, Miroslav, Mikolaj, Michiel, Naod, Peter, Paul, Razvan, Raphael, Rudolf, Souley, Stavros, Tobias, and Wibi. My Former collegues at FEIOG: Boy, Jeroen, Auke and Derk, thanks for taking me in and letting me go when I decided to pursue a PhD degree.

I also like to thank the people at ESnet for making me feel welcome away from home, I had a great time in both Berkeley and Champaign. Nick, Dop, Sam, Inder, Chin, Patty, Susan, Eli, Bruce, Eric, John, Jon, Chris, Kelly, Rebekah, Paul thank you for all your help and support. Sam, Dop and Jeanette thank you for inviting me into your homes and to various social events during my stay in Champaign.

During my travels abroad I interacted with many people that may have inspired my work. I like express gratitude to the many people in industry and in both NREN and scientific communities that I discussed my work with at conferences, side-meetings, and visits.

Of course the PhD life needs to be balanced with having fun with my long time friends, Rene, Suzan, Melda, Rox, Max, Fabian, Flip, Florian, Daniel, Steve, Sanne, Marieke and Daphne and the new ones that I made in the past years that have been very supportive: Stef, Rob, Carin, Peter, Arjen, Akke, Marije, Laurens, Kitty, KJ, Kelly, Andreea. Tristan, Pieter, JP, Cedric, Wietze and the rest of #bofhblaat thanks for the laughs, banter, gossip and useful technical discussions that occasionally distracted me from writing. Peter, Selmar, Roel, Marcel, Danny, thanks for the necessary distraction every Tuesday and for (almost) landing sticks on my head when thinking too much about work. Rian's colleagues at Studiemeesters, thank you for showing your interest and for understanding.

Also my family requires some kind words for their support: Henk, Hanneke dank voor jullie steun, nieuwsgierigheid en interesse. Oma, Lisette, Carlo, Ceejay en Jaylinn, dank voor het geduld en het begrip voor de keren dat ik weer eens ergens niet bij kon zijn omdat ik weer eens in het buitenland was. Hans, Josje, Adri, Lia, Theo en anderen die het proces van een afstand gevolgd hebben, erg bedankt voor jullie interesse.

Last but not least, I like to thank my cats Spookje, Jason, Shadow and Nibbler for all the typos they introduced in my thesis by walking over the keyboard when they tried to comfort me during the writing process. You contributed in interesting ways.

I realise the list of people who supported me during the last four years is overwhelmingly long. I tried to capture everyone, but forgive me if you are not mentioned and if you feel that your name should have been there.

SUMMARY

In this dissertation, we show that it is possible to build Secure Autonomous Response NETworks (SARNETs) that protect their infrastructure against attacks. Software Defined Networking, Network Function Virtualisation and cloud technologies allow us to run services in virtual networks as an overlay on top of cloud infrastructures. SARNET adds control software that uses control loops to monitor the virtual infrastructure and the health of the components that it supports. Various metrics of the service and the network are monitored using observables, which, together, safeguard the security state of the network. By monitoring groups of observables we can identify and classify attacks and start automated defences within the domain itself or multi domain in an alliance.

To experiment with SARNETs and to determine the basic defensive actions that can be provided by a software defined environment, we developed a testbed environment VNET. The environment emulates internet services inside an overlay network on existing cloud infrastructure in which we can safely attack these services using software. Using a graphical interface with visualisations of the attack we gathered input from experts during SuperComputing 2015 on how to construct automated defences.

The SARNET-agent is the control software that automatically defends the SARNET. The agent uses a control loop that constantly analyses the defined observables in the SARNET that runs in the VNET environment. Using the expert input from our discussions at SC15 we built defences that are automatically executed when an attack occurs, showing that it is possible to defend automatically. We evaluate the performance of the defences using the metrics *impact* and *efficiency* and we show how *efficiency* can be used to improve defence selection for subsequent attacks.

To precisely classify attacks we need to observe data coming from various data sources. We developed software, CoreFlow, that aids in this process by cross-referencing security events with recorded information from other systems in ESnet. In this case, we combined security events with network flow information and with topology information. We demonstrated that by having a combined view of these data sources we can extract new information such as the path that the attack traffic takes in the network and possible entry points.

This new information can help to mitigate attack traffic at the entry points of the single domain network.

To experiment with attack mitigation beyond the domain border, we extended the testbed for multi-domain experiments and enabled the SARNET-agent to orchestrate multi-domain defences. Using this extended testbed, we identified some of the factors that play a role when developing multi-domain defences. We compare three defences that execute the same defensive tasks in a different order which can be used in a collaboration between networks, that we call an alliance. We measure their efficiency when defending against the same attack and analyse their differences.

Since trust is the foundation for successful collaboration, we developed a defence approach based on trust provided by the Social Computational Trust model. This model has three components based on historical interactions between the collaborators. We evaluated this trust-based approach against the previous approaches in situations where the quality of collaboration differs between members.

With the increased availability of software defined infrastructures, SARNETs are a feasible way to protect networks. This dissertation shows that a SARNET can defend itself automatically to, at least, a selection of attacks. Extrapolating from the autonomous defence capabilities and by providing the SARNET interconnection patterns, SARNET can facilitate the compartmentalisation that is needed to secure the future Internet.

SAMENVATTING

In dit proefschrift laten we zien dat het mogelijk is om Secure Autonomous Response NETworks (SARNET's) te bouwen die hun infrastructuur beschermen tegen aanvallen. Software Defined Networking, Network Function Virtualization en cloudtechnologieën stellen ons in staat om diensten in virtuele netwerken te bieden als een overlay bovenop bestaande cloudinfrastructuren. SARNET voegt besturingssoftware toe die besturingslussen gebruikt om de virtuele infrastructuur en de gezondheid van de verschillende componenten en diensten binnen de infrastructuur te bewaken. Verschillende statistieken van de diensten en het netwerk worden bewaakt met behulp van observabelen die samen de beveiligingsstatus van het netwerk bewaken. Door groepen van deze te observeren, kunnen we aanvallen identificeren en classificeren en geautomatiseerde verdedigingen starten binnen een enkel domein of in meerdere domeinen die deel uitmaken van een alliantie.

Om te experimenteren met SARNETs en om de initiele defensieve acties die worden geleverd door een door software gedefinieerde omgeving te bepalen hebben we een testbedomgeving ontwikkeld die we VNET noemen. De omgeving emuleert internetdiensten binnen een overlay-netwerk op bestaande cloudinfrastructuur en zorgt ervoor dat we onze eigen diensten kunnen aanvallen zonder gevaar te vormen voor andere internet gebruikers.

Via een grafische interface met visualisaties van de aanval hebben we input van experts verzameld tijdens SuperComputing 2015 over het bouwen van geautomatiseerde verdedigingen. De SARNET-agent is de besturingssoftware die automatisch het SARNET verdedigt. De agent gebruikt een regellus die constant de observabelen in het SARNET, die wordt uitgevoerd in de VNET-omgeving, analyseert. Door inbreng van experts uit onze discussies op SC15 te gebruiken hebben we verdedigingen weten te creëren die automatisch uitgevoerd worden wanneer een aanval plaatsvindt. Hieruit blijkt dat het mogelijk is om automatisch te verdedigen. We evalueren de prestaties van deze verdedigingen door gebruik te maken van de door ons gedefineerde metrieken *impact* en *efficiency* en we laten zien hoe *efficiency* kan worden gebruikt om de verdedigingsselectie voor toekomstige aanvallen te verbeteren.

Om aanvallen precies te classificeren, moeten we gegevens observeren die afkomstig zijn van verschillende gegevensbronnen. We

hebben software ontwikkeld, CoreFlow, die dit proces helpt door de beveiligingsgebeurtenissen te combineren met informatie die is opgeslagen in andere systemen op het netwerk van ESnet. In dit geval hebben we de beveiligingsgebeurtenissen gecombineerd met netwerkverkeersinformatie en met netwerktopologie-informatie. We hebben de werking van CoreFlow aangetoond door met de door CoreFlow gecombineerde informatie nieuwe informatie te creëren: het pad dat het aanvalsverkeer in het netwerk volgt en de mogelijke toegangspunten waardoor het aanvalsverkeer binnenkomt. Deze nieuwe informatie kan helpen om aanvalsverkeer op de toegangspunten van het netwerkdomein te beperken.

Om te experimenteren met aanvalsbestrijding buiten de domeingrens hebben we het testbed uitgebreid om multi-domein experimenten uit te voeren en geven we de SARNET-agent de capaciteiten om multi-domein verdedigingen te organiseren. Met behulp van dit uitgebreide testbed hebben we enkele factoren geïdentificeerd die een rol spelen bij de ontwikkeling van verdedigingen met meerdere domeinen. We vergelijken drie verdedigingen, die dezelfde verdedigende taken uitvoeren in een andere volgorde, die kunnen worden gebruikt binnen een samenwerking van netwerken die wij een alliantie noemen. We meten de efficiëntie van de verschillende methodes die dezelfde aanval verdedigen en analyseren hun verschillen.

Omdat vertrouwen de basis is voor succesvolle samenwerking, hebben we een verdedigingsmethode ontwikkeld die gebaseerd is op vertrouwen die voortkomt uit het Social Computational Trust model. Dit model heeft drie componenten die gebaseerd zijn op de historisch interacties tussen de samenwerkende partijen. We hebben deze op vertrouwen gebaseerde aanpak vergeleken met de eerder genoemde aanpakken in situaties waar de kwaliteit van samenwerking verschilt tussen de leden.

Door de verhoogde beschikbaarheid van softwaregedefinieerde infrastructuren zijn SARNET's een haalbare manier om netwerken te beschermen. Dit proefschrift laat zien dat een SARNET zichzelf automatisch kan verdedigen tegen een selectie van aanvallen. Door middel van de autonome verdediging en door het aanbieden van de SARNET-verbindingspatronen, kan SARNET de compartimentering vergemakkelijken die nodig is om het toekomstige internet te beveiligen.