

# Measuring the effectiveness of SDN mitigations against cyber attacks

Ralph Koning, Ben de Graaff, Robert Meijer, Cees de Laat, Paola Grosso  
System and Network Engineering group (SNE) University of Amsterdam, The Netherlands  
Email: r.koning@uva.nl, b.degraaff@uva.nl, r.j.meijer@uva.nl, delaat@uva.nl, p.grosso@uva.nl

**Abstract**—To address increasing problems caused by cyber attacks, we leverage Software Defined networks and Network Function Virtualisation governed by a SARNET-agent to enable autonomous response and attack mitigation. A Secure Autonomous Response Network (SARNET) uses a control loop to constantly assess the security state of the network by means of observables. Using a prototype we introduce the metrics *impact* and *effectiveness* and show how they can be used to compare and evaluate countermeasures. These metrics become building blocks for self learning SARNET which exhibit true autonomous response.

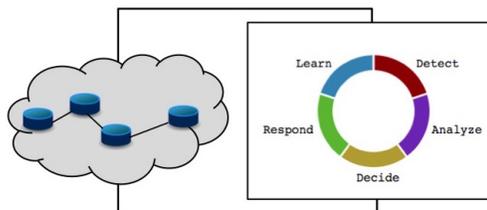


Fig. 1. The SARNET control loop

## I. INTRODUCTION

Computer networks are constantly being attacked. Cyber crime directed to network infrastructures and network protocols is increasing. The economic and societal consequences of such attacks are reaching front pages in the news leading to diminished trust in the Internet. In the era of the new Software Defined Networks (SDN), the crucial and interesting question is: To which level, can we rely on software based solutions for providing defence services? In this paper we will use our architecture for Secure Autonomous Response Networks (SARNET) [1]. We will show how SDN-based countermeasures can be adopted for 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 effectiveness. In this article we will, therefore, lay the foundation for a standardised manner to define and measure effectiveness of SDN-based cyber attack mitigation measures.

## II. SECURE AUTONOMOUS RESPONSE NETWORKS

In the SARNET project we are researching how to enable autonomy of network response to attacks. SDN-based mitigation techniques are one of the essential components in this vision, as they allow to create networks that are able to autonomously respond and recovery when attacked. A SARNET uses control loops to monitor and maintain the desired state required by the security observables.

The SARNET control loop traverses the following steps:

**Detect** – the default state of a SARNET during normal operation. Whenever the SARNET detects an anomaly on the network it triggers the control loop.

**Analyse** analyses the characteristics of the particular attack. *Analyse* determines where the attacks originate,

which path they take in the network and what the target is.

**Decide** – evaluates past decisions and policies and determines the suitable countermeasure for the attack.

**Respond** executes the countermeasure.

**Learn** – stores data containing results and execution parameters for future reference.

### A. Attack detection and analysis

Several techniques exist to detect known attacks. The first technique relies on intrusion detection systems. Flow analysis is another established way of detecting anomalies in the network. Finally, machine learning can be applied for attack detection, as presented by Sommer et. al [2]. In all cases, false positives can be reduced by correlating events in the dataset to events from other detection methods. These events can be collected and correlated in Security Information and Event Management (SIEM) systems or correlated using an attack correlation pipeline such as CoreFlow [3].

Existing methods, such as the one described in [4], can be used to classify an attack. The author proposes to use a cascading chain of elements to formally describe an attack, starting from the tools used by the attackers, the vulnerability they exploit, the action they perform, the intended target and the results they accomplish. This approach seems promising and we will investigate its suitability in the SARNET context. When the attack is classified, the exact characteristics of the attack need to be analysed. *Analyse* obtains the additional information 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.

## B. Decide

*Decide* looks at the cost and effectiveness of the possible reactions. To make a decision *Decide* takes the following aspects into account: attack class; attack characteristics; risk of applying the countermeasure; knowledge of the network; costs of executing responses and effectiveness of the countermeasure in similar situations (previous results from *Learn*)

Effective reaction depends on the flexibility of the SARNET under attack, e.g. whether the SARNET is redundant or multi-homed, and depends on the location in the network to apply the countermeasures. In some cases machines or network elements can be added and link capacity can be increased. Dynamically changing link properties are 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.

## C. React and Learn

Software defined networks give the required flexibility required for SARNET to change traffic flows and re-route important traffic away from overloaded parts of the network towards other parts dedicated to traffic analysis. Combining the flexibility of SDNs with both Network Function Virtualisation and machine virtualisation is an even more powerful solution. Service Function Chaining (SFC), an emerging standard for network control plane operations [5], provides a suitable solution to connect these NFVs together.

The *Learn* step records the effect of the chosen actions. The data recorded by *learn* can be used to respond more quickly to similar attacks in the future. When the attack characteristics and its effectiveness values are recorded and learned by an algorithm they will be used next time to optimise the *Respond* phase. Nevertheless, it may be desirable to override the automatic execution of a specific countermeasure from the ones recorded previously. Therefore, we provide a way to override learned behaviour and implement a self defined response during *Respond*.

## III. TOWARDS AN ESTIMATE OF EFFECTIVENESS

Given a system like SARNET the interesting part is to determine the effectiveness of countermeasures. When focusing on the time dimension of a countermeasure we define three main intervals:

- 1) The time to detect,  $t_d$ , is the time from the moment the attack starts ( $t_{sa}$ ) until the moment the attack is detected ( $t_{thr-up}$ ), that is the time when the service metrics threshold(s) is crossed:  $t_d = t_{thr-up} - t_{sa}$ .
- 2) The time to implement,  $t_i$ , is the time elapsed from the moment the attack is detected until the moment the implementation of the countermeasure is completed ( $t_{cm-impl}$ ):  $t_i = t_{cm-impl} - t_{thr-up}$ .
- 3) The time to recover,  $t_r$ , 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 ( $t_{thr-down}$ ) threshold is recovered:  $t_r = t_{thr-down} - t_{cm,impl}$ .

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 *Analyze* 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.

Effectiveness of a countermeasure is given by taking the sum of the normalized impact of the attack and the normalized costs of the reaction to it.

The impact of an attack can be seen as the integral of the lost revenue between the detection time and the recovery time. Fig.2 shows a simplified graphical representation of this concept.

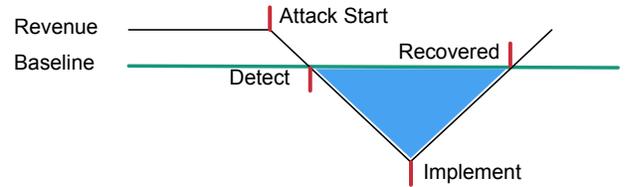


Fig. 2. Impact: the amount of lost revenue between the recovery time and the detection time (blue)

Once the thresholds are passed at the detection time the revenue might continue to decrease until the moment the countermeasures are in place; after that time the revenue starts to move toward the baseline until it reaches full recovery at the recovery time. The shape of the loss of revenue depends on the attacks characteristics.

The cost of a reaction can be determined using a similar approach. If cost is the integral of the (monetary) investments made to counter attack the same reasoning can be applied, yet cost increases in contrast to e.g. revenue that decreases when an attack occurs.

Two possible countermeasures are then comparable by looking at their respective values in terms of impact and costs. The solution with the lowest values is the most effective.

In realistic scenarios the effectiveness evaluation might be more complicated than just described. For instance, it is possible that even after the implementation of countermeasures there is no full recovery. In that case one could decide to finetune or change the response, until again the recovery is achieved. However, there are cases in which the thresholds will not be passed again, and thus the system will not fully recover. Even in these cases our effectiveness metric can be used to compare countermeasures if one also considers the difference between maximum recovery and the full recovery. The effectiveness considerations are not relevant purely for our SARNET architecture; the results are generalizable 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.

#### IV. THE SARNET PROTOTYPE

To evaluate our framework we further developed our VNET prototype [6]. VNET provides an orchestration and visualisation system for a SARNET which we currently deploy as an overlay network. It displays network topology information, flows and application metrics in an intuitive way. Additionally, it allows the creation of observables based on the current state of the network. The major components of VNET are depicted in Fig. 3. The *Infrastructure controller* talks to the IaaS platform to instantiate the virtual infrastructure, in this case ExoGENI [7]. 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; while *UI controller* and *VNET visualisation UI* display the network information and handle user interactions with VNET.

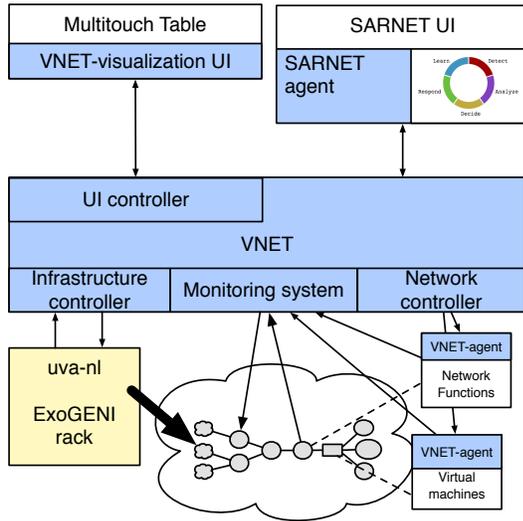


Fig. 3. Software components in the VNET prototype.

We introduced a number of new elements:

- support for virtual network functions and the infrastructure (SDN switch and a NFV host) to create VNFs that perform certain countermeasures.
- support 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<sup>1</sup> and subsequently sent to the VNET monitoring system.
- the SARNET-agent (Sec. IV-D) 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 SARNET-agent

the information and the tools it requires for autonomous network defence.

##### A. 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, ebttables<sup>2</sup> 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 virtual network function, cluster, or host, for processing. After processing the packets can then 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.

##### B. Network function virtualisation

The network function virtualisation host is currently implemented as a Linux host with a number of Docker<sup>3</sup> containers. Each container implements a specific network function. A Docker Registry instance is used to store a catalog of container images. All containers on the NFV host are attached to a Linux bridge. Using ebttables 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.

##### C. virtual network functions

Three different containers were made to run on the Docker host: an intrusion detection system (IDS), a CAPTCHA injector, 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 the proof of concept *all* clients are fully automated, only non-malicious clients can solve the challenge. 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

<sup>1</sup>host-sflow: <https://github.com/sflow/host-sflow>

<sup>2</sup>ebttables: <http://ebttables.netfilter.org>

<sup>3</sup>docker: <http://www.docker.io>

brute force attack, the honeypot can capture information on the accounts being attacked and the passwords being tried.

#### D. SARNET-agent

The SARNET-agent implements the SARNET control loop described in Sec. II. To show the state of the SARNET-agent and the information it uses to make its decisions we use an extra visualisation UI besides the one that is provided by VNET. With this visualization we present various network metrics such as network flows and total bandwidth usage. We show application metrics such as CPU usage, transaction rate, and successful versus failed login attempts; and we display 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.

### V. 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 and password attack.

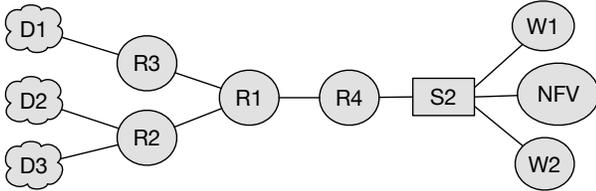


Fig. 4. Topology of the virtual network: Three domains ( $D1-D3$ ) are connected via multiple routers ( $R1-R4$ ) and a switch ( $S2$ ) to two web services ( $W1-W2$ ).  $NFV$  is a host that runs our security VNFs.

Fig. 4 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  $S2$  switch described in the previous section. Under normal circumstances simulated users in the network domains  $D1-D3$  send regular requests to the web services  $W1-W2$ . The amount of successful requests will generate the *Revenue* value we use in our measurements. In our attack scenarios, attacks originate from the external domains  $D1-D3$  and target the web services  $W1-W2$ .

In the UDP attack 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. The SARNET-agent recognises the type of attack due to the excessive amount of UDP traffic the simultaneous drop in revenue. The SARNET has two possible countermeasures to apply: increasing the bandwidth of the core links or filtering the malicious traffic at the edges (routers  $R2-R3$ ).

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

servers is increased. The increase in turn affects the capability to answer legitimate requests. Since these resource requests happen at the application layer, the network layer will not clearly show indication of an attack. Therefore, SARNET will first deploy 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 second step, it redirects all requests from the domains where the bad traffic originates, i.e. IP ranges, to a container running a CAPTCHA. In this case, as countermeasure, the traffic is redirected by  $S2$  to the NFV host  $NFV$  which in this case has started both the IDS and CAPTCHA VNFs. After filling in the CAPTCHA regular traffic is redirected to the web servers while the automated malicious traffic is blocked.

In the Password attack malicious users are trying to log in on the servers by attempting logins with dictionary generated passwords. This again takes place on the application layer. In this case the SARNET again responds by first deploying an IDS on the  $NFV$  host to identify the attackers in  $D1$ . However, in this case, the SARNET starts a honeypot VNF and unlike the CPU attack scenario, the SARNET-agent now uses the intel gathered from the IDS to let the SDN switch  $S2$  only redirect the *identified* malicious users to the honeypot that is deployed dynamically on the container host.

Now that the attackers are routed to the honeypot, the web servers  $W1-W2$  can resume normal operations.

### VI. 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. The time to detect the attack is purely dependent on the amount of attack traffic. This is to be expected since the time when the alarm is triggered, i.e.. the time in which the threshold is passed, occurs at an earlier time as the attack has a stronger incept. What we can see is that among various network runs there is very little variability among this detection time: this means that as long as the threshold is well tuned to the desired sensitivity level the time to identify that an attack is occurring, will be fairly constant.

When we instead look at the recovery time in this same scenario we have the first indication that the type of software-defined response we apply in the overlay network has an influence. Fig. 5 presents this time for the two types of responses we had implemented, namely the increase of the available bandwidth in the core links or the application of filters at the edges close to the attackers. In the first case (rate change) we observe that at a certain point there is no recovery possible, indicated in the figure with the missing boxplot. This means that 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.

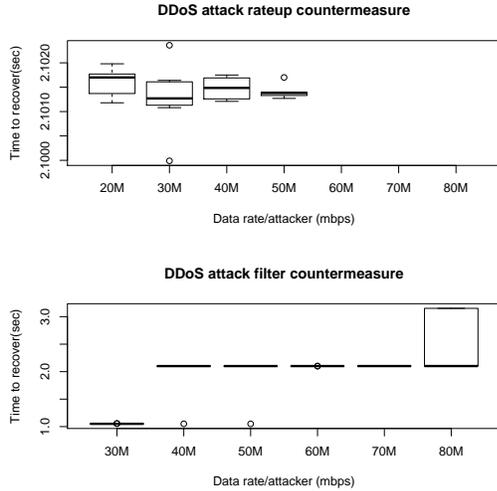


Fig. 5. Time to recover after the implementation of a countermeasure (in seconds) as a function of each individual attacker UDP rate. Top plot shows the results when applying a rate increase in the core; bottom plot refers to the application of filters

In the CPU attack scenario we simulate a varying the 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.

The implementation of the countermeasure has two steps: first we deploy an IDS to classify the requests and secondly we redirect all suspicious connections to 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 function of the number of attackers. Fig. 6 shows that the recovery time goes from an average of 6.55 seconds for 3 attackers to 23.5 seconds when there are 15 malicious nodes. This can be explained by observing that a larger number of attackers will bring the amount of successful transactions much further down from the threshold, consequently it will take longer to reach and pass the threshold again once the countermeasure is in place.

When we analysed the performance of our system under a password attack we see that the detection time is independent of the number of attackers. 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 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 is continuously updates as failed logins occur.

The time the system takes to recover after the successful implementation of the countermeasures in a password attack has no dependency on the number of attackers. This is because the redirect to the honeypot happens instantly.

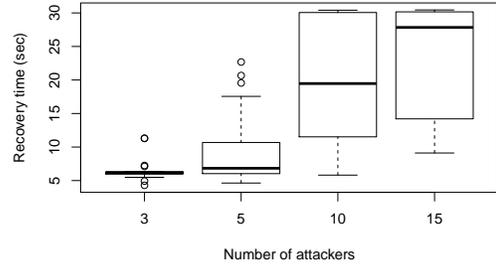


Fig. 6. Time to recover from a CPU attack (in seconds) as function of the number of attackers.

## VII. DISCUSSION

There are three main elements that affect the impact, as it can be seen from Fig.2 and derived from our results. Firstly, the thresholds set to identify attacks will determine the time at which we start to evaluate the integral; second, the scale and characteristics of the attacks themselves might influence the shape of the revenue curve in time; and finally, the measures that are used to safeguard the network will determine the value of the implementation time and the recovery time.

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.

The approach to determine effectiveness is crucial when deciding how to respond to an attack. As we had shown in Fig.1 our system comprised a Learn phase that will store effectiveness information and use at subsequent time to take the most appropriate decisions.

## VIII. RELATED WORK

Defence mechanisms against network attacks have been thoroughly compared against each other in the literature. In particular approaches for the mitigation of DDoS attacks have received significant attention. Surveys have been conducted, for example by Chang et al. [8] or more recently by Zargar et al. [9]. These surveys provide an extensive evaluation of various techniques but they do not provide quantitative ways to define effectiveness as we do in this paper. Such definitions are crucial to support the learning and decision making required an autonomously reacting systems, and our approach provides that.

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. [10] address these aspects, and point to the need to extensive evaluation of SDN-based solutions and SDN networks themselves. We believe that our proposal to evaluate countermeasures by effectiveness, will facilitate the assessment of software based responses.

Existing work so far has mainly focused on the survey of VNFs techniques and discussing their applicability in various scenarios, particularly in data centres [11] and mobile environments [12] [13]. 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 [14](2010) that machine learning could be successfully applied to the area of intrusion detection. Recent patents such as the one from Google on botnet detection [15] show the applicability of this type approach for identifying attacks. Our intent is to use machine learning to assess effectiveness and adopt the most effective set of countermeasures.

## IX. CONCLUSIONS AND FUTURE WORK

This paper shows the first steps toward autonomous response to cyber attacks using SDN and NFV. We introduce the SARNET control loop, elaborated on the phases of the control loop and discussed how to implement them. We also showed a first implementation of this control loop as a continuation of the VNET work, which after including SDN and NFV capabilities was able to exhibit autonomous response to a selection of attacks. We define impact and effectiveness and show how these metrics can be used to evaluate different solutions. Finally, our measurements show that detection and response time are dependent on the attacks characteristics and we argue that though we don't see variation in implementation time, it will increase and vary when complex risk assessments are required. We conclude that metrics for impact of the attack and effectiveness of a countermeasure are necessary inputs for learning and choosing the best suitable responses to achieve more advanced autonomous responses.

The actual assessment of relative effectiveness is the focus of our future work. We are interested in using our evaluation system to compare multiple SDN measures and to select the best option, and on determining where to apply such measures in the network when there are multiple options. Furthermore, we think that containers have the potential to share security VNFs such as detection mechanisms, and possible countermeasures in a reusable manner. Therefore, we want to continue to investigate intelligence sharing using containers in multi domain collaborations such as SARNET Alliances [16].

## ACKNOWLEDGEMENTS

SARNET is funded by the Dutch Science Foundation NWO (grant no: CYBSEC.14.003 / 618.001.016) and the National project COMMIT (WP20.11) Special thanks to CIENA for

hosting our demonstration at their booth at SC16 and in particular, Rodney Wilson, Marc Lyonais, and Gauravdeep Shami for providing feedback on our work and their continuous support. We also thank our other research partners TNO and KLM.

## REFERENCES

- [1] R. Koning, A. Deljoo, S. Trajanovski, de Graaff, Ben, 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," *OFC The Optical Networking and Communication Conference 2017 - under submission*, 2017.
- [2] 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.
- [3] R. Koning, N. Buraglio, C. de Laat, and P. Grosso, "Coreflow: Enriching bro security events using network traffic monitoring data," *Innovating the Network for Data Intensive Science (INDIS) workshop at Super Computing 2016, Salt Lake City (UT)*, 2016.
- [4] J. D. Howard and T. A. Longstaff, "A common language for computer security incidents," *Sandia National Laboratories*, 1998.
- [5] P. Quinn and T. Nadeau, "Service function chaining problem statement," *IEEF RFC 7498*, 2015.
- [6] 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)*, June 2016, pp. 483–488.
- [7] 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.
- [8] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the dos and ddos problems," *ACM Comput. Surv.*, vol. 39, no. 1, Apr. 2007. [Online]. Available: <http://doi.acm.org/10.1145/1216370.1216373>
- [9] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 2046–2069, Fourth 2013.
- [10] 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," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 602–622, Firstquarter 2016.
- [11] L. R. Battula, "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)*, Sept 2014, pp. 1622–1628.
- [12] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc)," *IEEE Network*, vol. 28, no. 6, pp. 18–26, Nov 2014.
- [13] M. Chen, Y. Qian, S. Mao, W. Tang, and X. Yang, "Software-defined mobile networks security," *Mobile Networks and Applications*, vol. 21, no. 5, pp. 729–743, 2016. [Online]. Available: <http://dx.doi.org/10.1007/s11036-015-0665-5>
- [14] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE Symposium on Security and Privacy*, May 2010, pp. 305–316.
- [15] S. Ranjan, J. Robinson, and F. Chen, "Machine learning based botnet detection using real-time connectivity graph based traffic features," Jun. 24 2014, uS Patent 8,762,298. [Online]. Available: <https://www.google.com/patents/US8762298>
- [16] A. Deljoo, L. Gommans, C. de Laat, T. van Engers et al., "The service provider group framework," *Looking Beyond the Internet: Workshop on Software-defined Infrastructure and Software-defined Exchanges*, 2016.