Future Generation Computer Systems 91 (2019) 144–156

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# Measuring the efficiency of SDN mitigations against attacks on computer infrastructures

R. Koning [*], B. de Graaff, G. Polevoy, R. Meijer, C. de Laat, P. Grosso

*System and Network Engineering Group (SNE), University of Amsterdam, The Netherlands*

## HIGHLIGHTS

- SARNET maintains a secure network state using control loops and security observables.
- Metrics for impact an efficiency are important for comparing countermeasures.
- Ranking countermeasures by efficiency can assist in making defence decisions.

## ARTICLE INFO

*Article history:*
Received 30 January 2018
Received in revised form 13 June 2018
Accepted 7 August 2018
Available online 20 August 2018

*Keywords:*
Software Defined Networks
Network Function Virtualization
Cyber attacks
Cyber security
Defence efficiency
Overlay networks

## ABSTRACT

Software Defined Networks (SDN) and Network Function Virtualisation (NFV) provide the basis for autonomous response and mitigation against attacks on networked computer infrastructures. We propose a new framework that uses SDNs and NFV to achieve this goal: Secure Autonomous Response Network (SARNET). In a SARNET, an agent running a control loop constantly assesses the security state of the network by means of observables. The agent reacts to and resolves security problems, while learning from its previous decisions. Two main metrics govern the decision process in a SARNET: *impact* and *efficiency*; these metrics can be used to compare and evaluate countermeasures and are the building blocks for self-learning SARNETs that exhibit autonomous response. In this paper we present the software implementation of the SARNET framework, evaluate it in a real-life network and discuss the tradeoffs between parameters used by the SARNET agent and the efficiency of its actions.

## 1. Introduction

Crime directed to network infrastructures and network protocols is increasing [1]. The economic and societal consequences of such attacks are reaching front pages in the news leading society to question their trust in the Internet [2–4]. Not surprisingly, an entire industry emerged to create an ecosystem of tools and devices that are marketed to prevent, stop, or to mitigate the negative effects of such malicious behaviour. We can install off the shelf Intrusion Detection Systems (IDS) to identify the existence of attacks and we can deploy specialised firewalls to prevent malicious traffic from entering a specific network domain.

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 naturally is whether SDNs can provide novel methods to counteract attacks.

Another emerging technology in computer networking is Network Function Virtualisation (NFV). NFV allows the instantiation and placement of Virtual Network Functions (VNF) in the network on the fly [5]. 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, traffic scrubbing facilities can all be deployed flexibly where most needed.

We are convinced that SDNs and VNFs are suitable for attack response mechanisms. In case of attacks on network infrastructure, SDNs and VNFs bring 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 paper we will use our architecture for Secure Autonomous Response Networks (SARNET) [6]. We will show how SDN-based

* Corresponding author.
*E-mail addresses:* r.koning@uva.nl (R. Koning), b.degraaff@uva.nl (B. de Graaf), g.polevoy@uva.nl (G. Polevoy), r.j.meijer@uva.nl (R. Meijer), delaat@uva.nl (C. de Laat), p.grosso@uva.nl (P. Grosso).

https://doi.org/10.1016/j.future.2018.08.011
0167-739X/© 2018 Elsevier B.V. All rights reserved.

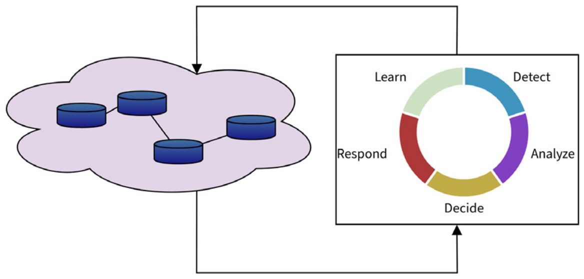


**Fig. 1.** The SARNET control loop.

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 efficiency. In this article we will, therefore, lay the foundation for a generic manner to define and measure the efficiency of SDN-based mitigations against computer infrastructures. The impact and efficiency metrics presented in this paper 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.

The rest of this article is organised as follows: Section 2 presents the conceptual operation in a SARNET and Section 3 discusses the impact and efficiency metrics. Section 4 describes the software prototype that implements a SARNET and Section 5 lists the scenarios in which we have tested its performance. Sections 6, 7, and 8 present and discuss the experimental setup and the results of our evaluation. Finally, we frame our efforts in the context of existing work in Section 9 and summarise in Section 10 the current status and future steps towards SARNET adoption in real-life networks.

## 2. Secure autonomous response networks

Nowadays, software can efficiently support the instantiation of network topologies as an overlay network on physical devices. Virtual switches, virtual links and virtual network functions, together, are the building blocks for software-defined overlay networks. Companies increasingly rely on overlay networks for both the delivery of services to their customers, or for the establishment of inter-company services. An example is the creation of virtual networks between instances of cloud based virtual machines or containers. While these virtual networks are technically feasible their robustness during attacks has not yet fully evaluated. Novel approaches to both the detection of attacks as well as the implementation of defence strategies are key elements to achieve sufficient robustness.

In the SARNET project we are researching how to ultimately enable autonomy of network response to attacks. SDN-based techniques are promising components in this vision as they provide the flexibility and means to autonomously deploy countermeasures when attacked. A SARNET uses control loops to monitor and maintain the desired state required by the security observables. The SARNET control loop is similar to the OODA loop (observe, orient, decide, and act). Lenders et al. [7] successfully applied the OODA loop to cyber security. The SARNET loop, shown in Fig. 1, has an added step, compared to the OODA loop, namely the *learn* phase. In this phase data on the attack characteristics as well as data on the defence adopted are collected and stored to improve response times during future attacks.

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.

The various steps in the control loop are carried out in the SARNET-agent component. This component receives information from one or more external monitoring systems for *Detect*; it relies on a network controller for the execution of the *Respond* stage. The SARNET-agent, monitoring system and network controller work closely together to maintain the network's security state.

### 2.1. Attack detection and analysis

Several techniques exist to detect known attacks. The first technique relies on intrusion detection systems; these systems can, when updated regularly, detect most known attacks. Flow analysis is another established way of detecting anomalies in the network. 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 applied for attack detection. Sommer et al. [8] researched the use of machine learning in intrusion detection systems and identified some challenges. They stated that Machine learning is much better at detecting similarities than detecting outliers. To use machine learning one needs to train the algorithm with network data during under normal operation as well as during attacks. The latter dataset is often difficult to obtain since this requires to collect facets of network behaviour during such anomalous events. Furthermore, even if one manages to train the algorithm with sufficient data, there is a risk of registering false positives, so further investigation by a security expert is necessary when the network under attack is critical to the business. 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 the one we developed for CoreFlow [9].

Existing methods, such as the one described in [10], 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.

### 2.2. Decide

*Decide* looks at the cost and efficiency 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
- Efficiency 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.

### 2.3. React and learn

SDNs provide the 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 NFV and machine virtualisation enables deployment of countermeasures where required. Service Function Chaining (SFC), an emerging standard for network control plane operations [11], 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. deep packet inspection, filtering, or sanitation. Exclusively processing suspicious flows lowers the cost of the response and is less disruptive to regular traffic. Once the reaction is in place, the network evaluates whether or not the applied countermeasure has the desired effect. The *Decide* step in the next run will evaluate whether the countermeasure is still required and sufficient. It will take care of initiating removal or applying an additional or new defence based on updated information from the *Learn* step.

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. 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 difference between revenue recovered thanks to the reaction and cost of the reaction itself. We will elaborate on this efficiency definition in Section 3. What constitutes an effective countermeasure depends on this efficiency metric but will differ between SARNETs because of differences in network topology, rules and policies. When the attack characteristics and efficiency 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*.

## 3. Towards an estimate of efficiency

Given a system like SARNET, determining the efficiency of countermeasures is crucial to estimate how well the system functions and to learn how to automatically apply the best response. Prior to formally defining efficiency, we define a system 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

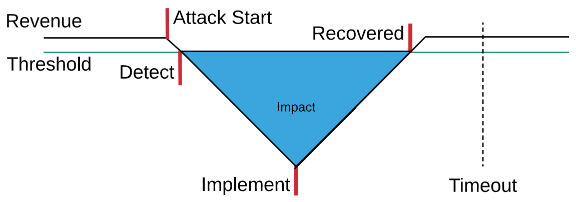


**Fig. 2.** Here, a recovery takes place. Impact: the amount of the lost revenue between the detection time and the recovery time (blue area). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

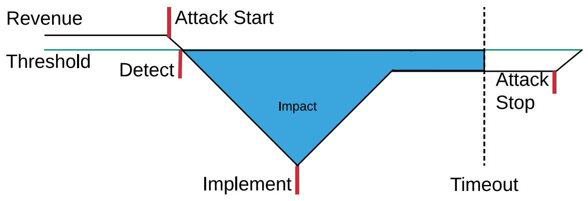


**Fig. 3.** No recovery takes place. Impact: the amount of the lost revenue between the detection time and the end of the time window.

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 in the following 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.

### 3.1. Impact

The impact of an attack can be defined with respect to the chosen observable, such as the 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 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. 2 shows a simplified graphical representation of this concept, when a recovery takes place. Fig. 3 illustrates a case without recovery.

The moment at which the threshold is passed 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 ultimately fully recovers, defining the recovery time, or never recovers, at least not 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 the revenue when it is below the threshold. The revenue is lower-bounded at zero, as we cannot have a negative revenue; consequently, we do not need to specially 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. Implementing such a ceiling causes, as a side effect, the impossibility to distinguish which countermeasure performs worse, if the revenue repeatedly exceeds the ceiling. 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 pass 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.

## 3.2. *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 by machine learning, 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 till the recovery, or until the end of the time window, if no recovery has taken place till then. 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 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 $E\colon \{\text{recovered, not recovered}\} \times \mathbb{R}_+ \times \mathbb{R}_+ \to [0, 1]$.

From the infinitely many definitions of efficiency that fulfil all the above properties, we propose the following one. We define the efficiency as

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

where parameter $\beta$ defines the cutoff between recovery and no recovery (we allocate $\beta$ 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 make a compelling argument for this efficiency function, we strengthen the above requirements and prove that the stronger set of requirements actually characterises Eq. (1).

**Theorem 1.** *Let $Ct$ obtain values in $[0, C \cdot T]$. Then, Eq. (1) is the unique definition of efficiency that satisfies the following set of properties:*

1. *Linearly decreasing with impact $I$, where $I \in [0, B \cdot T]$.*
2. *Linearly decreasing with total cost $Ct$, where $Ct \in [0, C \cdot T]$.*
3. *The ratio of the linear coefficient of the impact to the linear coefficient of the total cost is the same, regardless whether the recovery takes place or not.*
4. *If no recovery takes place, all the values between 0 and $\beta$ and only they can be obtained; if a recovery does take place, then all the values between $\beta$ and 1 and only they can be obtained.*

We remark that condition 3 implies that the ratio of the linear coefficient of the impact to the linear coefficient of the total cost expresses their relative importance, regardless whether recovery takes place.

**Proof.** Eq. (1) is linearly decreasing with impact and with total cost and condition 3 holds in a straight-forward manner. We have shown after the definition of Eq. (1) that condition 4 is fulfilled as well. It remains to prove the other direction.

Let the formula for the case when *a recovery is attained* be $a - b \cdot I - d \cdot Ct$, for positive $b$ and $d$. This form follows from conditions 1 and 2. For the minimum impact and total cost, $I = Ct = 0$, we have the maximum possible efficiency of 1, implying that $a - b0 - d0 = 1 \Rightarrow a = 1$. For the maximum impact and total cost, $I = B \cdot T$ and $Ct = C \cdot T$, we have the minimum possible efficiency of $\beta$, which means that $1 - b \cdot BT - d \cdot CT = \beta \Rightarrow bBT + dCT = 1 - \beta$. Let $\alpha$ be $bBT$. The nonnegativity of $dCT$ and $bBT + dCT = 1 - \beta$ imply together that $bBT \leq 1 - \beta$, as required from $\alpha$ in Eq. (1). Moreover, $bBT + dCT = 1 - \beta$ implies that $dCT = 1 - \beta - \alpha$. To conclude, the efficiency is $1 - b \cdot I - d \cdot Ct$, where $b = \frac{\alpha}{BT}$ and $d = \frac{1-\beta-\alpha}{CT}$, for $\alpha \in [0, 1-\beta]$, as in Eq. (1).

In the case of *no recovery*, let the formula be $a' - b' \cdot I - d' \cdot Ct$. By substituting $I = Ct = 0$ we conclude that $a' = \beta$. By substituting $I = BT$ and $Ct = CT$, we obtain $\beta - b'BT - d'CT = 0$, i.e. $b'BT + d'CT = \beta$. From condition 3 we have

$$\frac{b}{d} = \frac{b'}{d'} \iff \frac{b}{b'} = \frac{d}{d'}.$$

These two equations, together with the proven above equality $bBT + dCT = 1 - \beta$, imply that each coefficient gets multiplied by $\frac{\beta}{1-\beta}$, yielding $b' = b\frac{\beta}{1-\beta}$ and $d' = d\frac{\beta}{1-\beta}$. Together with the expression above for $a'$, we obtain Eq. (1). □

Since in our model we assume that there are no costs for applying countermeasures, we will omit the total cost part by using $\alpha = 1 - \beta$. Instead of directly calculating the efficiency of the non-recovered runs, we use the success rates from Table 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): $E_m(\text{Recovered}, I) \triangleq 1 - \frac{I}{B \cdot T}$, 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 $E_{\text{SARNET}}$ by the success rate. Here, the nonnegative parameter $\gamma_i$ describes the importance of $i$th revenue. By taking normalised $\gamma_i$s, such that the combination is convex, meaning that $\sum_{i=1}^{n} \gamma_i = 1$, we ensure that $E$ is in [0, 1], because all the $E_i$s are there.

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. This requires the system to store the full time data of the impact per an observable of each attack. Therefore, it is important to set the threshold and the ceiling carefully before the measurements commence.

We have suggested a natural efficiency function that is characterised by a set of reasonable properties, and then we have simplified it for our usage. Therefore, these efficiency considerations are not relevant purely 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. 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, flows and application metrics in an intuitive way. Additionally, it allows the creation of observables based on the current state of the network.

In our previous paper [12] we described in detail the major components of VNET as depicted in Fig. 4. Here we provide a short summary thereof:

- *Infrastructure controller* talks to the IaaS platform to instantiate the virtual infrastructure; in our case, we use ExoGENI [13] a cloud platform that provides good network level isolation.
- *Monitoring system* receives monitoring information from the virtual infrastructure.
- *Network controller* controls the network and hosts in the virtual infrastructure.
- *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.
- *UI controller* and *VNET visualisation UI* display the network information and handle user interactions with VNET.

For autonomous defence we developed a SARNET-agent (Section 4.4) 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

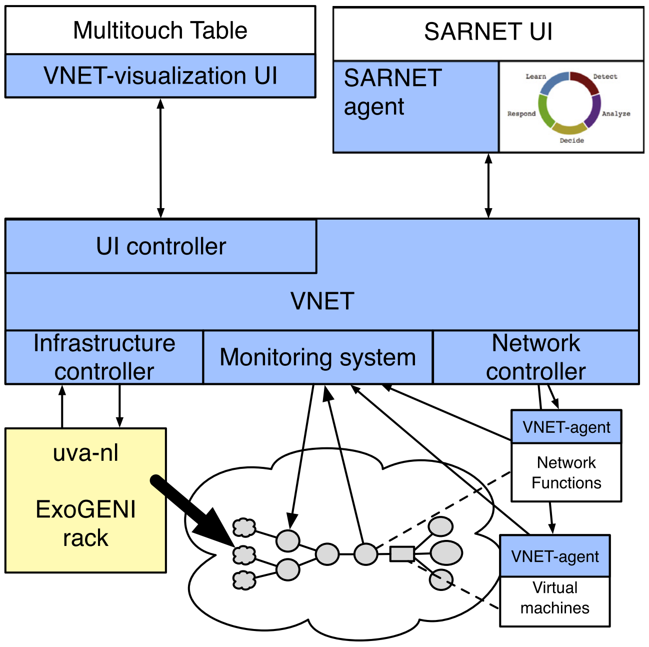


**Fig. 4.** Software components in the VNET prototype.

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 components in more detail.

### 4.1. 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 CAPTCHA by using cookies that only non-malicious clients set.

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.

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

### 4.2. 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 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.

### 4.3. 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. 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.

### 4.4. SARNET-agent

The SARNET-agent implements the SARNET control loop described in Section 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.

[2] ebtables: http://ebtables.netfilter.org.

[3] docker: http://www.docker.io.

### 4.5. 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. 5) besides the one that is provided by VNET. 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.

## 5. 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

Fig. 6 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* containing a mix of high and low resource pages as well as correct and incorrect logins 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 asynchronously collected by the SARNET-agent with a sample rate of approximately 1 s. From these metrics we define the following observables that are monitored for health:

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

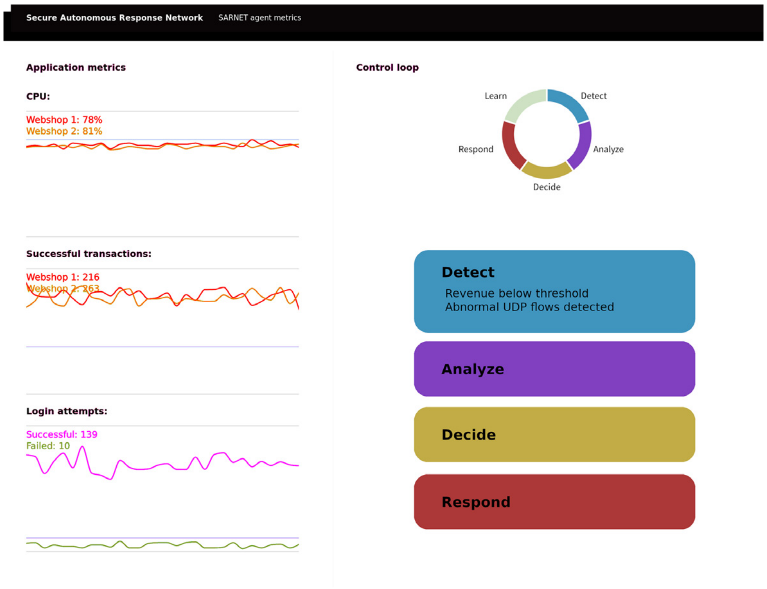
When one of these observables fails the SARNET-Agent launches the associated countermeasure.

### 5.1. UDP 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 towards 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 increase the bandwidth

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


Secure Autonomous Response Network SARNET agent metrics

Application metrics

CPU:

Webshop 1: 78%
Webshop 2: 81%

Successful transactions:

Webshop 1: 216
Webshop 2: 263

Login attempts:

Successful: 139
Failed: 10

Control loop

Detect
Revenue below threshold
Abnormal UDP flows detected

Analyze

Decide

Respond

**Fig. 5.** Top right part of user interface of the SARNET-agent, it visualises the metrics the agent uses, the control loop and the decisions taken.

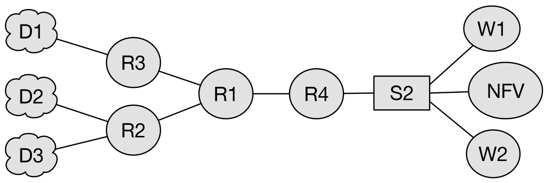


**Fig. 6.** Topology of the virtual network: Three domains (*D1*–*D3*) are connected via multiple routers (*R1*–*R4*) and a switch (*S2*) to two web services (*W1*–*W2*). *NFV* is a host that runs our security VNFs.

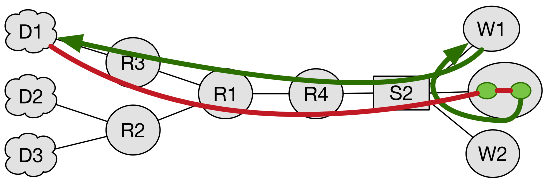


**Fig. 7.** The mixed (red) traffic (attack + normal requests) from *D1* is redirected to the NFV host which has two VNFs chained, first an IDS that monitors the traffic, finally an CAPTCHA blocker that prevents malicious requests to pass and normal traffic (green) to continue to web services (*W1*–*W2*). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

of the core links using the *tc* traffic control utility; in the latter we filter the malicious traffic at the edges (routers *R2*–*R3*) using *iptables*.

### 5.2. 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 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 computational 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 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.

Fig. 7 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.

### 5.3. 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.

As can be seen in Fig. 8, 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

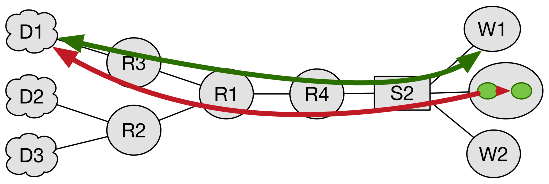


**Fig. 8.** The mixed (red) traffic (attack + normal requests) from *D1* is redirected to the NFV host which has two VNFs chained, first an IDS that monitors the traffic, finally a honeypot that can monitor attack behaviour. In this case normal requests (green) pass through untouched to (*W1*–*W2*). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

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.

## 6. Test setup

To evaluate whether 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. 6. Each time we start a new scenario, we reset the virtual network to the default state and wait for the network to stabilise.

Section 5 described the attack scenarios: DDoS, CPU, and password attack and the four countermeasures used 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 s; these correspond to increase of 2, 3, and 4 times the window size of 10 s. 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 20 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 Section 3; 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.

## 7. Simulation results

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

### 7.1. Time evolution of the SARNET

It is illustrative to 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. Figs. 9 and 10 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 show the detection times, the implementation times implement1 and implement2, and the start 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. 9 a DDoS attack is mitigated and the sales climb back up above the set threshold after the implementation of the countermeasure.

Fig. 10 shows an unsuccessful mitigation of a password attack. After recording three samples where the number of logins exceeds 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 does not fall back below the acceptable value within the allotted time.

For the CPU attack recovery needs to happen on multiple thresholds, namely revenue and CPU load. Figs. 11 and 12 illustrates two runs in which the systems does not recover. In the first case the sales do not pass again the set threshold, while the CPU load does; in the second case both metrics do not fall back within the acceptable range.

### 7.2. 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 high impact attacks. This characterisation is specific to each attack. For DDoS we define light as 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 them light when we have a scenario with 5 attackers, medium with 10 attackers and heavy with 15 attackers.

Table 1 list 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 s by default.

Success rate indicates whether countermeasures are suitable against a specific attack. As we can read from Table 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 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

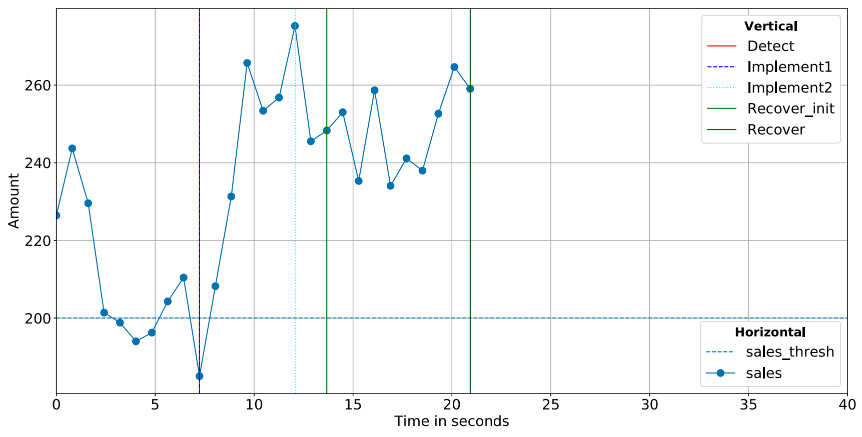


**Fig. 9.** Successful run with one threshold (DDoS attack). Note: The implement1 line is plotted on top of the detect line since the events occurred at the same time.

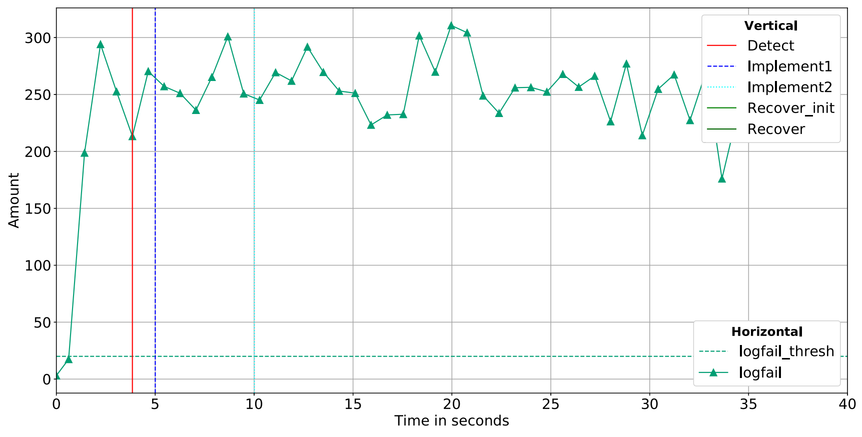


**Fig. 10.** Failed run one threshold (password attack).

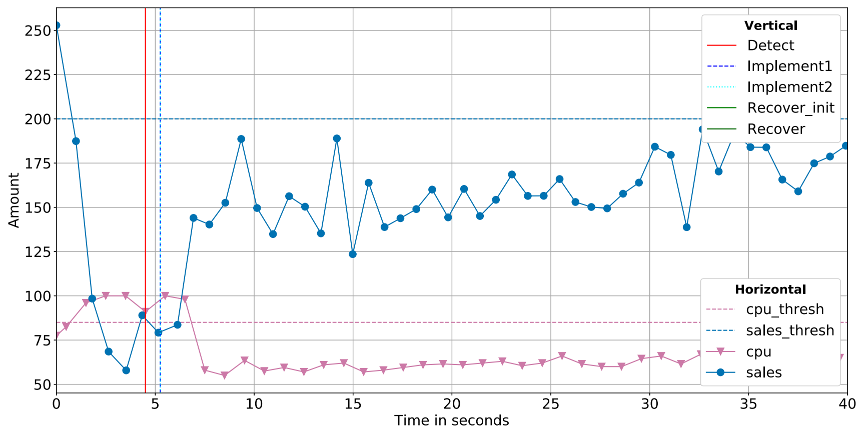


**Fig. 11.** Failed run with only one threshold recovered other not (CPU attack).

not affect the recovery time. There is a 1 s fluctuation which is close to the interval at which we sample the metrics (0.8 s).

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 s (or 2, 3, and 4 times the window size of 10 s) during a Medium sized attack. Table 3 shows the success rate of the experiments; as expected 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 Section 3 there are cases in 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 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.

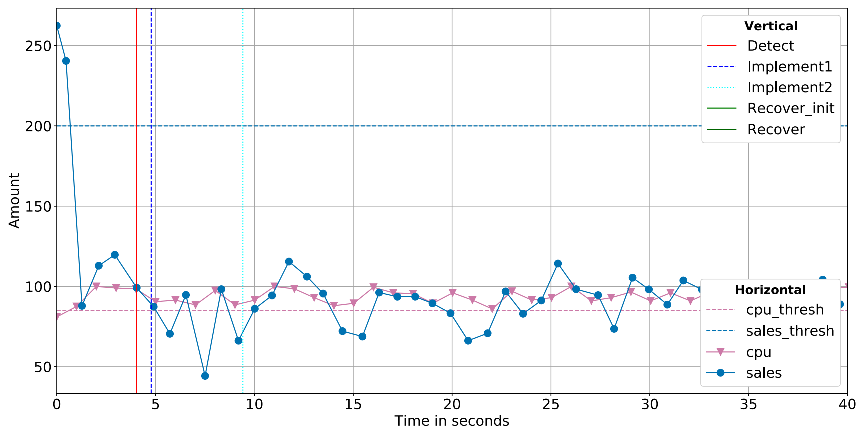


**Fig. 12.** Failed run where both thresholds do not recover (CPU attack).

**Table 1**
Success ratio of recovery for the various attacks intensities as function of the applied countermeasure.

| | Size | % attacks recovered | | |
|---|---|---|---|---|
| **Attack** | **Defence** | Light | Medium | Heavy |
| **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 2**
Recovery time for successful runs for the various attacks intensities as function of the suitable countermeasures.

| | Attack size | Recovery time (in seconds) | | |
|---|---|---|---|---|
| **Attack** | **Defence** | Light | Medium | Heavy |
| **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 |

**Table 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.

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

**Table 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 s.

| **Attack** | **Defence** | 0% | 5% | 10% | 15% |
|---|---|---|---|---|---|
| **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% |

**Table 5**
Impact of the countermeasures for the various attacks intensities as function of the applied countermeasure.

| | | Attack size | Impact | | |
|---|---|---|---|---|---|
| **Attack** | **Defence** | **Type** | Light | Medium | Heavy |
| **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 |

## 7.3. Impact and efficiency

Section 3 showed how we determine impact and efficiency of various countermeasure to an attack. Table 5 reports on the impact of the attack as function of the size of the attack. Not surprisingly, we see that the impact of the attack on the system 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 Heavy attack. This is due to the artificial ceiling (2x baseline) that we used as a maximum to keep the impact of each measurement within a range. This procedure was described in Section 3.1. When we remove this limit, the values for login failures give us the expected increase.

The effect of the ceiling is not an issue when comparing two different responses within the same attack category; on the other hand, it is not possible anymore to rank the effect of the same countermeasures for various attack intensity, as the ceiling makes us rank all the countermeasures as equally good.

Table 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; 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 6**
Efficiency of the countermeasures for the various attacks intensities as function of the applied countermeasure.

| | Size | Efficiency | | | Efficiency × Success rate | | | |
|---|---|---|---|---|---|---|---|---|
| | | Light | Medium | Heavy | Light | Medium | Heavy | |
| **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.

## 8. 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 make sense within the same attack class. Besides attack class there other factors that influence efficiency:

- the thresholds set to identify attack;
- the time spent on risk analysis 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 characteristics of the attacks;
- and finally the execution time of the selected countermeasure.

Because the configuration of the SARNET sets the thresholds and timeouts and the risk analysis and decision are common for all countermeasures, the only variables changing is the attack scale and characteristics. To get a good measurement for the countermeasure efficiency, the attacks scale and characteristics needs to be constrained. In this paper we used three categories, Light, Medium and Heavy. Table 6 showed that there are indeed different values for efficiency as the scale changes; this implies for example that the efficiency of a countermeasure during a light attack is not necessarily representative when under heavy attack; generally a heavy attack has less effective countermeasures because of limited resources (e.g. bandwidth).

In Section 7 we mentioned that the artificial ceiling we use to limit excessive values skewed some measurements. Currently, we made this dependent on the threshold by limiting the values to a maximum of $2 \times threshold$, this scales the maximum with the expected value of the metric. Normalising by the maximum amount may give a more sensible image but to compare to new individual runs the number has to be the same across runs. This limitation also applies to the threshold; one can only compare effectiveness of the runs with the same threshold set. In environments with dynamic thresholds, e.g. self learning, or based on time of day, one has to keep the individual data points of all runs and recompute effectiveness of the runs one wants to compare to.

The countermeasure analysis in this paper was done after completing all runs. However, the goal is to perform such an analysis after each run and update the ranking and average measurements immediately. This increases the accuracy of the average after each measurement. Eventually, the best solution will be picked first all the time, leaving limited or no experience with subsequent solutions or new solutions that have no efficiency metric yet. This can be solved by forcing new solutions to be tried first, however this is not always desirable in a production environment since running unknown, potentially impacting, or sub-optimal solutions first will negatively impact the restoration time of the service. Keeping the time limited to execute a new countermeasure and immediately backing it up with the top ranked defence when the new countermeasure fails can however be an acceptable strategy. Another approach is to first test the effects and efficiency of the countermeasure in a similar staging environment and to implement it later in the production system initialised with the metrics from the staging environment.

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

## 9. Related work

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

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. [14] or more recently by Zargar et al. [15]. These surveys provide an extensive evaluation of various techniques but they do not provide quantitative ways to define efficiency 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.

Granadillo et al. [16] describe how countermeasures can be ranked using the RORI index [17] which includes several factors, such as infrastructure costs, risk assessment and attack surface. Our paper focuses on a subset of the factors considered in RORI. Instead of using an estimative approach our ranking is based on empirical data on how well a countermeasure 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. [18] address these aspects, and point to the need of extensive evaluation of SDN-based solutions and SDN networks themselves. We believe that our proposal to evaluate countermeasures by efficiency will facilitate the assessment 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 [19] and mobile environments [20,21]. Previous work has often relied on simulation to assess SDN use as mitigation to attacks, e.g. in the work of Wang et al. [22]. 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 [23](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 [24] show the applicability of this type approach for identifying attacks. Our ultimate goal of using machine learning to assess efficiency and adopt the most effective set of countermeasures is, therefore, a novel and promising application of such techniques.

## 10. Conclusions and future work

This paper shows the first steps towards 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 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 countermeasures implemented on SARNET and showed how this approach allows us to rank countermeasures based on efficiency.

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

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.

A first next step is to include cost in our impact and efficiency evaluation. Afterwards we plan to build a learning system based on such efficiency metric; this will allow us to automatically update the ranking of countermeasures every time new attacks occur such that a SARNET can ultimately exhibit efficient recovery.

Finally, we showed that is it possible to develop and deploy countermeasures as containers. We believe that containers have the potential to be used for sharing security VNFs such as detection mechanisms, and other possible countermeasures in a reusable manner. Therefore, our current effort is to investigate container based intelligence sharing in multi domain collaborations such as SARNET Alliances [25].

## Acknowledgments

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 Lyonnais, and Gauravdeep Shami for providing feedback on our work and their continuous support. We also thank our other research partners TNO and KLM.

## References

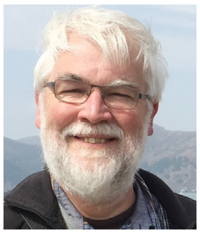
[1] M. Jonker, A. King, J. Krupp, C. Rossow, A. Sperotto, A. Dainotti, Millions of targets under attack: a macroscopic characterization of the DoS ecosystem, in: Proceedings of the 2017 Internet Measurement Conference, ACM, 2017, pp. 100–113.
[2] L. Rainie, J. Anderson, The fate of online trust in the next decade, URL http://www.pewinternet.org/2017/08/10/the-fate-of-online-trust-in-the-next-decade.
[3] Internet Society, ISOC Global Internet Report 2017, 2017, URL https://www.internetsociety.org/globalinternetreport/2017/.
[4] World Economic Forum, Global Risks Report 2018, 2018, URL https://www.weforum.org/reports/the-global-risks-report-2018.
[5] B. Han, V. Gopalakrishnan, L. Ji, S. Lee, Network function virtualization: Challenges and opportunities for innovations, IEEE Commun. Mag. 53 (2) (2015) 90–97, http://dx.doi.org/10.1109/MCOM.2015.7045396.
[6] R. Koning, A. Deljoo, S. Trajanovski, de Graaff, Ben, P. Grosso, L. Gommans, T. van Engers, F. Fransen, R. Meijer, R. Wilson, C. de Laat, Enabling e-science applications with dynamic optical networks: Secure autonomous response networks, in: OFC The Optical Networking and Communication Conference 2017 - under submission.
[7] V. Lenders, A. Tanner, A. Blarer, Gaining an edge in cyberspace with advanced situational awareness, IEEE Secur. Priv. (2) (2015) 65–74.
[8] R. Sommer, 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.
[9] R. Koning, N. Buraglio, C. de Laat, 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.
[10] J.D. Howard, T.A. Longstaff, A Common Language for Computer Security Incidents, Sandia National Laboratories, 1998.
[11] P. Quinn, T. Nadeau, Service function chaining problem statement, IEEF RFC 7498, 2015.
[12] R. Koning, B. de Graaff, C. de Laat, R. Meijer, P. Grosso, Interactive analysis of SDN-driven defence against distributed denial of service attacks, in: 2016 IEEE NetSoft Conference and Workshops, NetSoft, 2016, pp. 483–488, http://dx.doi.org10.1109/NETSOFT.2016.7502489.
[13] I. Baldine, Y. Xin, A. Mandal, P. Ruth, C. Heerman, 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.
[14] T. Peng, C. Leckie, K. Ramamohanarao, Survey of network-based defense mechanisms countering the DoS and DDoS problems, ACM Comput. Surv. 39 (1) (2007), http://dx.doi.org/10.1145/1216370.1216373.
[15] S.T. Zargar, J. Joshi, D. Tipper, A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks, IEEE Commun. Surv. Tutor. 15 (4) (2013) 2046–2069, http://dx.doi.org/10.1109/SURV.2013.031413.00127.
[16] G.G. Granadillo, M. Belhaouane, H. Debar, G. Jacob, RORI-based countermeasure selection using the OrBAC formalism, Int. J. Inf. Secur. 13 (1) (2014) 63–79.
[17] N. Kheir, N. Cuppens-Boulahia, F. Cuppens, H. Debar, A service dependency model for cost-sensitive intrusion response, Comput. Secur. ESORICS 2010 (2010) 626–642.
[18] Q. Yan, F.R. Yu, Q. Gong, 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 Commun. Surv. Tutor. 18 (1) (2016) 602–622, http://dx.doi.org/10.1109/COMST.2015.2487361.
[19] 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, 2014, pp. 1622–1628, http://dx.doi.org/10.1109/ICACCI.2014.6968453.
[20] H. Hawilo, A. Shami, M. Mirahmadi, R. Asal, NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC), IEEE Netw. 28 (6) (2014) 18–26, http://dx.doi.org/10.1109/MNET.2014.6963800.
[21] M. Chen, Y. Qian, S. Mao, W. Tang, X. Yang, Software-defined mobile networks security, Mob. Netw. Appl. 21 (5) (2016) 729–743, http://dx.doi.org/10.1007/s11036-015-0665-5.
[22] B. Wang, Y. Zheng, W. Lou, Y. Hou, DDoS attack protection in the era of cloud computing and software-defined networking, Comput. Netw. 81 (2015) 308–319.
[23] R. Sommer, V. Paxson, Outside the closed world: on using machine learning for network intrusion detection, in: 2010 IEEE Symposium on Security and Privacy, 2010, pp. 305–316, http://dx.doi.org/10.1109/SP.2010.25.
[24] S. Ranjan, J. Robinson, F. Chen, Machine learning based botnet detection using real-time connectivity graph based traffic features, US Patent 8, 762, 298, Jun. 24 2014, URL https://www.google.com/patents/US8762298.
[25] 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.

**Ralph Koning** received his M.Sc. in System and Network Engineering in 2007 at the University of Amsterdam. After being employed in the System and Network Engineering research group at the UvA he started his Ph.D. in 2015 on the SARNET project. He contributed work to several projects such as GigaPort, CineGrid, GN3plus and COMMIT. His current interests include computer networks, SDN infrastructures, semantic descriptions and digital security. https://staff.fnwi.uva.nl/r.koning/.

**Ben de Graaff** is a software engineer with interests in network programming and security. He has received his M.Sc. in System and Network Engineering from the University of Amsterdam. He now works for the UvA a scientific programmer on the SARNET project. His main topics of interest are secure communication, real-time monitoring, and network/infrastructure automation.

**Gleb Polevoy** received his B.A. in mathematics and computer science and his M.Sc. degree in computer science from Technion, Israel Institute of Technology, Haifa, Israel. He received his Ph.D. from Delft University of Technology, Delft The Netherlands, and since then he has been a post-doctorate researcher in the SNE group at the University of Amsterdam. His research interests include game theory, social choice theory and approximation algorithms.

**Prof. Dr. Robert Meijer** (1959) has a Ph.D. of the University of Utrecht in experimental nuclear physics (1988). Then he developed particle detection systems at GSI Darmstadt and ISP's for the Dutch Telco KPN. In 2002 Robert became part-time professor 'Applied Sensor Networks' at the University of Amsterdam and developed a sensor lab and a global scale self-learning, -scaling anomaly detection systems for dikes. In 2016 Meijer developed the concepts for the digital transformation of the Metropole Region Rotterdam and The Hague. There showed that digital economies require first of all "normative systems"; ICT that observes other ICT to determine if they behave according to law, regulations and appointments. In 2017 he developed a concept that strongly mitigates cyber-security risks and that allows secure and norm-obeying transactions on data.



**Prof. de Laat** chairs the System and Network Engineering (SNE) laboratory in the Informatics Institute of the Faculty of Science at University of Amsterdam. The SNE lab conducts research on leading-edge computer systems of all scales, ranging from global-scale systems and networks to embedded devices. Across these multiple scales our particular interest is on extra-functional properties of systems, such as performance, programmability, productivity, security, trust, sustainability and, last but not least, the societal impact of emerging systems-related technologies. Prof. de Laat serves on the Lawrence Berkeley Laboratory Policy Board for ESnet, is co-founder of the Global Lambda Integrated Facility (GLIF), founder of GRIDforum.nl and founding member of CineGrid.org. His group is/was part of a.o. EU projects GN4-2, SWITCH, CYCLONE, ENVRIplus and ENVRI, Geysers, NOVI, NEXTGRID, EGEE, and national projects DL4LD, SARNET, COMMIT, GIGAport and VL-e. He is a member of the Advisory Board Internet Society Netherlands and Scientific technical advisory board of SURF Netherlands. See: http://delaat.net/.

**Dr. Paola Grosso** is associate professor in the System and Network Engineering (SNE) group at the University of Amsterdam. She is the coordinator and lead researcher of all the group activities in the field of multi-scale networks and systems. Her research interests lie in the creation of sustainable e-Infrastructures, relying on the provisioning and design of programmable networks. She co-chaired the NML-WG (Network Markup Language Working Group) within OGF (Open Grid Forum). She been a member of the SC organising committee for SCinet for 7 years. She has been involved in several FP-7 projects, namely NOVI, ENVRI, Geysers and MOTE. She currently participates in several national projects, such as SARNET, DL4LD, SecConNet and in EU H2020-funded projects such as FED4FIRE+, GN4 and ENVRIPLUS.