Measuring the Efficiency of SDN Mitigations Against Cyber Attacks

Ralph Koning[a,*], Ben de Graaff[a], Robert Meijer[a], Cees de Laat[a], Paola Grosso[a]

[a] *System and Network Engineering Group (SNE), Universiteit van Amsterdam, Science Park 904, Amsterdam, The Netherlands*

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 *efficiency* 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.

Keywords: SDN, Autonomous Respons, cyber attacks, security, Software Defined Networks, SARNET, NFV, Network Function Virtualization

1. 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. 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 behaviours. We can install off the shelf Intrusion Detection Systems to identify the existence of attacks and we can deploy specialised firewalls to prevent malicious traffic from entering a specific network domain.

However, in the era of the new Software Defined Networks (SDN), the following crucial and interesting question comes up: To which level, can we rely on software based solutions for providing defence services? Concretely we ask ourselves if there are novel ways to utilise SDNs and Virtual Network Functions (VNF) to counter attacks. If we can do this, is it possible to use Artificial Intelligence (AI) to coordinate the actions of these VNFs and SDNs in such a way that responses to attacks are autonomously taken, ultimately without external (human) intervention?

We are convinced that SDNs and VNFs are suitable for such response mechanisms. In this paper we will use our architecture for Secure Autonomous Response Networks (SARNET)[1]. We present the new components to our VNET[7] framework to enable automatic autonomous response. Using these components we apply SDN-based countermeasures that can be used for protection of networks and ultimately for guaranteed delivery of services. We argue that, in order to select the best countermeasure for an attack, the most useful element of our solution, or for that matter any other SDN-based network, is a proper characterisation of the countermeasure's efficiency. In this article we will, therefore, lay the foundation for a standardised manner to define and measure efficiency of SDN-based cyber attack mitigation measures.

*Corresponding author

Email addresses: r.koning@uva.nl (Ralph Koning), c.b.degraaff@uva.nl (Ben de Graaff), r.j.meijer@uva.nl (Robert Meijer), delaat@uva.nl (Cees de Laat), pgrosso@uva.nl (Paola Grosso)

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 elements 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 and the assessment thereof require novel approaches to both detection of attacks as well as the implementation of defence strategies.

In the SARNET project we are researching how to ultimately 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 is similar to the OODA loop (observe, orient, decide, and act). Lenders et al. successfully applied the OODA loop to cyber security [2]. The SARNET loop shown in Fig. 1 has an added step, compared to the OODA loop, which increases the granularity. In the explicitly added *learn* phase data is collected and stored to improve response times for 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

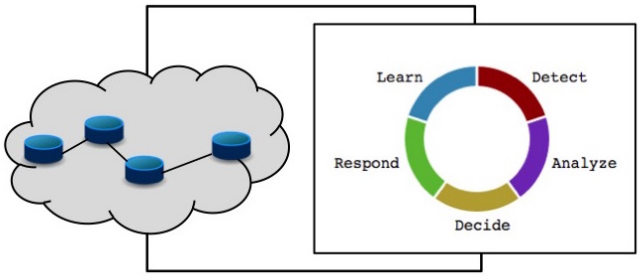

Figure 1: The SARNET control loop

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 [3] researched the use of machine learning in intrusion detection systems and note some challenges. The paper states that Machine learning is much better at detecting similarities than detecting outliers. To use use machine learning one requires one needs to train the algorithm with data from the network during under normal operation and during attacks. The latter dataset is often difficult to obtain since this is a collection of network behaviour during anomalous events. 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 CoreFlow[4].

Existing methods, such as the one described in [5], 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 multihomed, 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

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 [6], 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 *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 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. This paper elaborates on this measure of efficiency by quantifying impact, the performance degradation of the network from the moment of detection until the moment of recovery. What constitutes an effective countermeasure depends on this efficiency metric but will differ between SARNETs and furthermore depends on non-technical 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 which uses control loop mechanisms to counter attacks the interesting part is to determine the efficiency of countermeasures. We argue that efficiency needs to be evaluated as a complex value, which has dependencies on the *footprint* of the attack, as well as on the timings of the response. With the former we mean that even if an attack has a specific signature and is recognised as belonging to a known type, the efficiency of the countermeasure will depend on the specific characteristics of the attack. When focusing on the time dimension of a countermeasure we define three main intervals:

- Time to detect: t_d
  The *time to detect* is the time from the moment the attack starts (t_{sa}) until the moment the attack is detected (t_{thr-up}), that is the time when the service metrics threshold(s) is crossed. $t_d = t_{thr-up} - t_{sa}$.

- Time to implement: t_i
  The *time to implement* is the time elapsed from the moment the attack is detected until the moment the implementation of the countermeasure is completed ($t_{cm-impl}$).
  $t_i = t_{cm-impl} - t_{thr-up}$.

- Time to recover: t_r
  The *time to recover* is the time elapsed from the moment the countermeasure is implemented to the moment until the service metrics are recovered, and the threshold is passed in the other direction ($t_{thr-down}$).
  $t_r = t_{thr-down} - t_{cm_impl}$.

In terms of the control loop, t_d is the time it takes in the *Detect* phase from the moment there is a trigger to the moment the control loop moves to the next phase. t_i is the time that the control loop spends in the *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.

Once more, the efficiency 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.

4. The SARNET prototype

To perform our study we further developed our VNET prototype[7]. 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.

In our previous paper we described the major components of VNET as depicted in Fig. 2. We summarise:

- *Infrastructure controller* talks to the IaaS platform to instantiate the virtual infrastructure, in this case ExoGENI[8].
- *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 (Sec. 4.4) that receive 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.

In order to accommodate our automated defences and new attacks we updated VNET to 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[1] and subsequently sent to the VNET monitoring system. Additionally, we added virtual network functions and the infrastructure (SDN switch and a NFV host) to create VNFs that perform certain countermeasures. The next sections will describe these additional components in more detail.

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

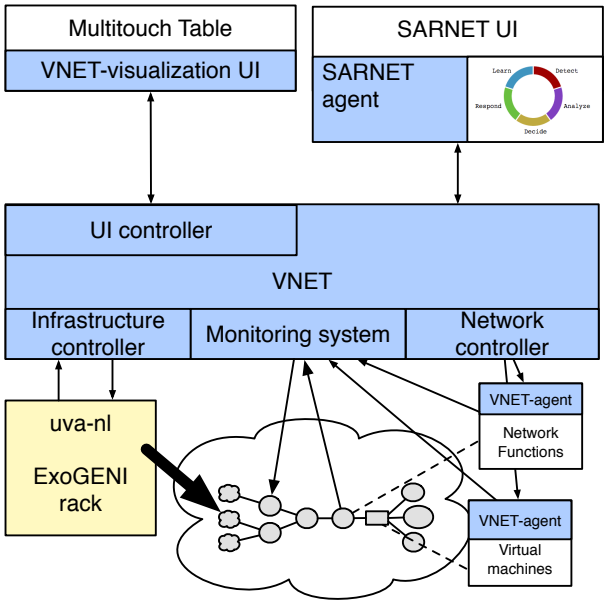

Figure 2: Software components in the VNET prototype.

4.1. 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 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.

4.2. Network function virtualisation

Network function virtualisation 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 catalog 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.

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

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

4.4. SARNET-agent

The SARNET-agent implements the SARNET control loop described in Sec. 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 *analyze* phase any changes in service and network state are processed. For example, service transactions per second, CPU usage, and the number of succesful 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 intrusion detection system, 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 intrusion detection system 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, rerouting traffic to the NFV host using an SDN switch, and choosing the chain of network functions to apply and to which IP addresses.

4.5. SARNET-agent UI

To show the state of the SARNET-agent and the information it uses to make its decisions we use an extra visualisation UI

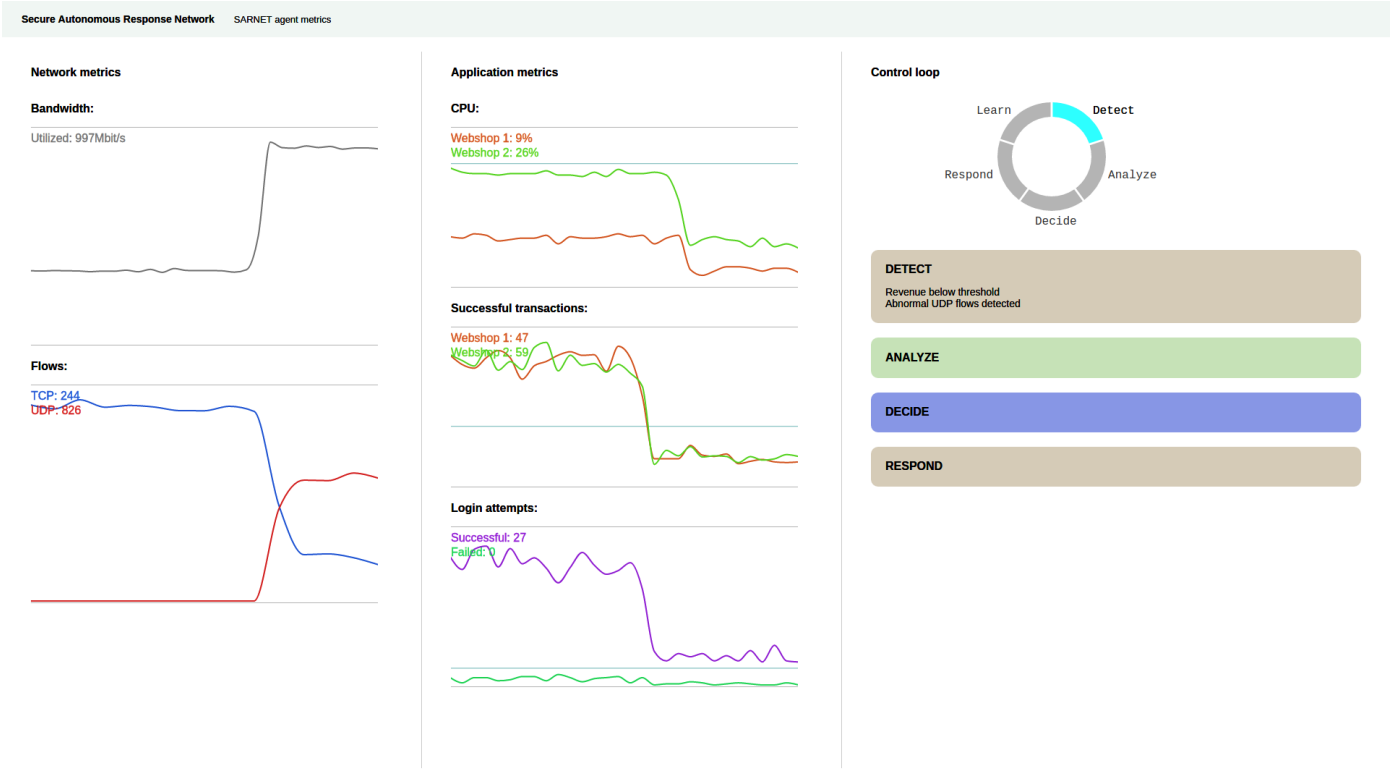

Figure 3: The user interface of the SARNET-agent, it visualises the metrics the agent uses, the control loop and the decisions taken. A video of the user interface is published at: `https://youtu.be/PDrRQkvIERo`.

(Fig. 3) besides the one that is provided by VNET. The first column 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. 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. 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*.

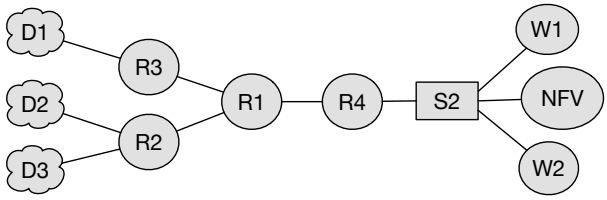

Figure 4: Topology of the virtual network: Three domains (*D1–D3*) are connected via multiple routers (*R1–R4*) and a switch (*S2*) to two web services (*W1–W2*). *NFV* is a host that runs our security VNFs.

This virtual network is under constant monitoring. We monitor the following metrics: 1) *revenue*, the amount of successful transactions to the web services, 2) *logfail*, the amount 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 asynchonously collected by the SARNET-agent with a sample rate of approximately 1 second. From these metrics we define the following observables that are monitored for health:

- ddos_observable; fails when the metric *revenue* 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 *revenue* passes their threshold

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

5.1. UDP attack

Here 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 and 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*). Sec. 6 will evaluate both countermeasures in to this attack scenario.

5.2. CPU utilisation attack

In this 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. Fig. 5 shows how the traffic is

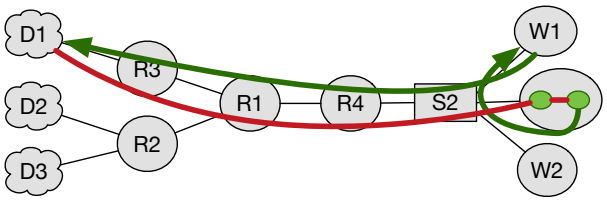

Figure 5: The mixed (red) traffic (attack + normal requests) from *D1* is redirected to the NFV host which has two VNFs chained, first an IDS that monitors the traffic, finally an CAPTCHA blocker that prevents malicious requests to pass and normal traffic (green) to continue to web services (*W1–W2*).

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.

5.3. Password attack

Here 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.

As can be seen in Fig. 6, 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*. 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*

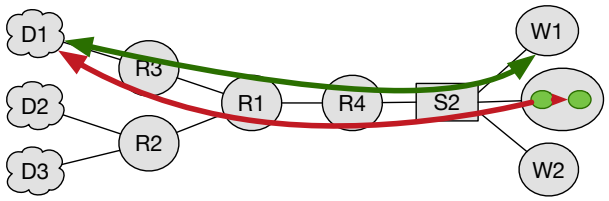

Figure 6: The mixed (red) traffic (attack + normal requests) from *D1* is redirected to the NFV host which has two VNFs chained, first an IDS that monitors the traffic, finally an honeypot that can monitor attack behaviour. In this case normal requests (green) pass through untouched to (*W1–W2*).

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. The honeypot gives the possibility to further analyse the passwords that the attackers use and to gain additional intelligence. Currently we do not use the honeypot intelligence to improve the SARNET detection systems; we consider this future work.

6. Results

As we have explained in Sec. 5 we have considered three types of attacks. In the following paragraphs we will present the results for the time to detect, time to implement and time to recover in the three scenarios.

6.1. UDP DDoS results

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

When we look at the recovery time this scenario we can see that the type of software-defined response we apply in the overlay network has an influence. Fig. 7 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.

6.2. CPU attack results

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, as it can be seen in Fig. 8.

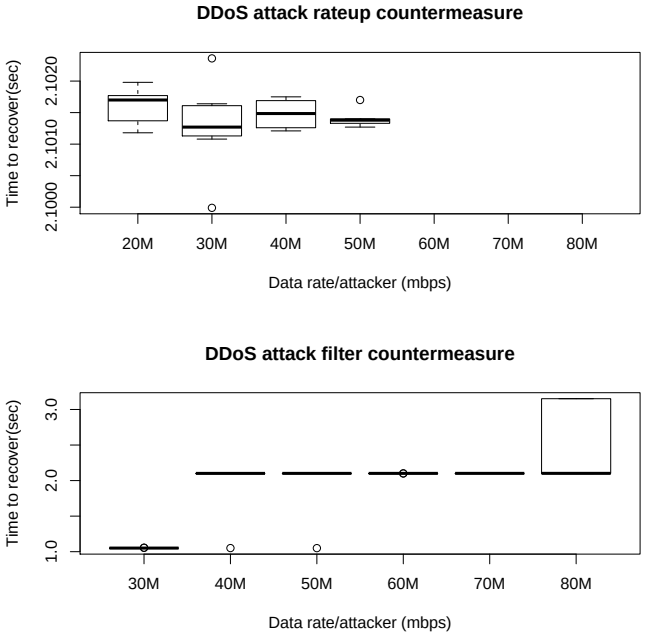

Figure 7: Time to recover after the implementation of a countermeasure (in seconds) as a function of each individual attacker UDP rate. Top plot shows the results when applying a rate increase in the core; bottom plot refers to the application of filters

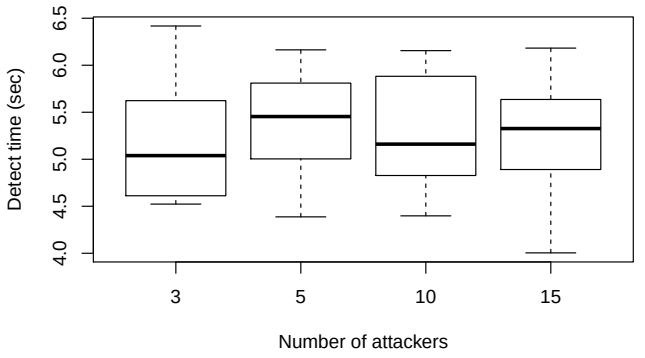

Figure 8: Time to detect a CPU attack (in seconds) as function of the number of attackers.

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. 9 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.

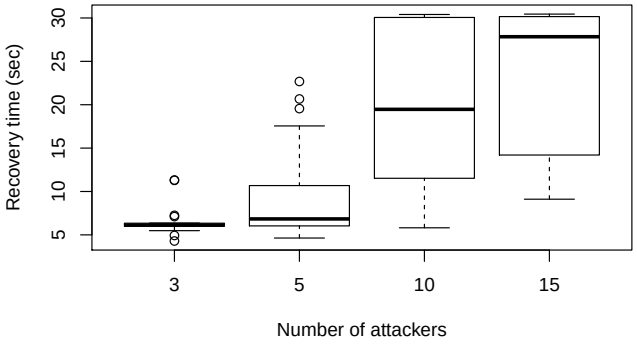

Figure 9: Time to recover from a CPU attack (in seconds) as function of the number of attackers.

6.3. Password attack results

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, as shown in Fig. 10. 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.

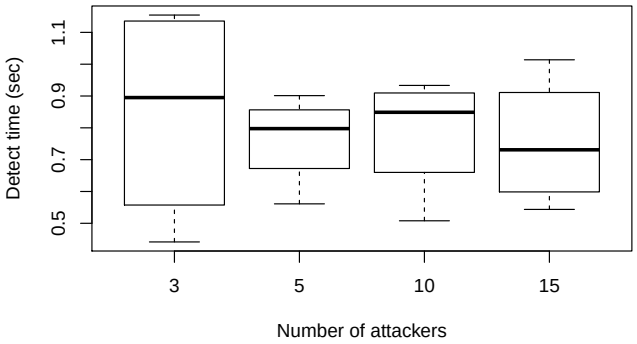

Figure 10: Time to detect a password attack (in seconds) as function of the number of attackers.

The implementation times of the two step defence are shown in Fig.11.

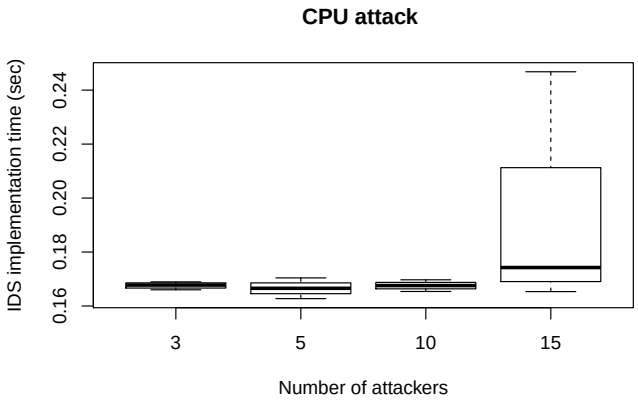

Figure 11: Time (in sec) to implement the two step defence in a CPU attack (top) and a password attack (bottom) as function of the number of attackers.

Fig.12 presents the time the system takes to recover after the successful implementation of the countermeasures in a password attack. In this case there is no dependency on the number of attackers. This is because the redirect to the honeypot happens instantly.

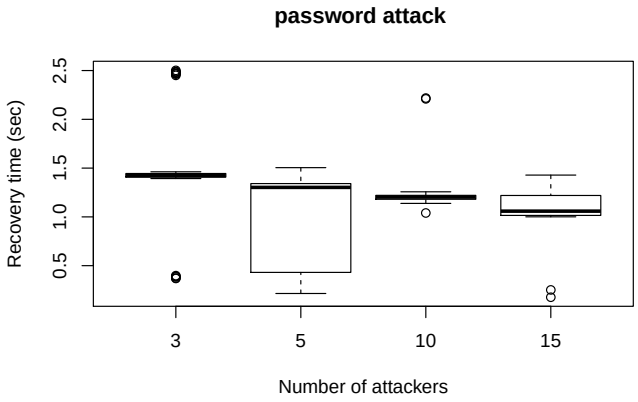

Figure 12: Time (in sec) to recover in a password attack as function of the number of attackers.

7. Discussion

Drawing from the results we now provide a way towards the assessment of the efficiency of countermeasures. Our generalised approach is meant to allow an easy comparison of different solutions in various implementations. This is particularly useful when assessing traditional, non SDN based solutions and comparing them to SDN-based solutions, as well as comparing SDN-based solutions to each other.

Efficiency of a countermeasure is given by taking the sum of impact of the attack and the costs of the reaction to it. The impact of an attack can be seen as the integral of one or more of the metrics described in Sec. 5. For example, we can take the integral of the *revenue* between the detection time and the recovery time and compare this to what the *revenue* should have been according to the baseline. This gives you the loss in *revenue* or the impact of the attack on the *revenue* metric. Fig.13 shows a simplified graphical representation of this concept. 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.

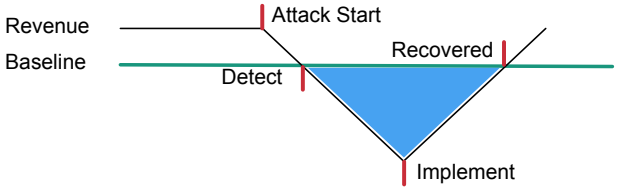

Figure 13: Impact: the amount of lost revenue between the recovery time and the detection time (blue)

The cost of a reaction is the integral of the (monetary) investments made to counter attack and the same reasoning made for the impact in terms of the shape of this function applies.

Two possible countermeasures are then comparable by looking at their performance values in terms of impact and costs and calculating the sum of these values. The solution with the lowest value is the more efficient.

In realistic scenarios the efficiency 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 such as depicted in Fig. 14.

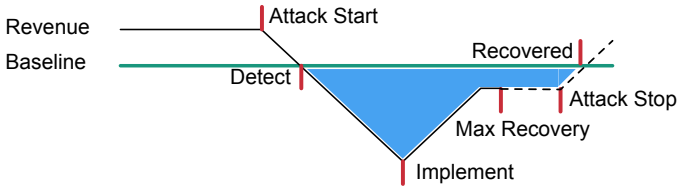

Figure 14: Impact: the amount of lost revenue between the recovery time and the maximum recovery (blue) in the case of a countermeasure that isn't capable of fully recovering. The delta between maximum recovery and the baseline can be used as a measure to select the best countermeasure when full recovery is not an option.

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 the system will be operating in suboptimal mode. Even in these cases our efficiency metric can be used to compare countermeasures if one also considers the difference between maximum recovery and the full recovery.

There are three main elements that affect the impact, as it can be seen from the plot and derived from our results:

- the thresholds set to identify attacks will determine the time at which we start to evaluate the integral;

- the scale and characteristics of the attacks themselves might influence the shape of the revenue curve in time;
- 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.

This approach to determine efficiency becomes 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 efficiency information and use at subsequent time to take the most appropriate decisions.

8. Related Work

Our work presents defence mechanisms against cyber attacks that rely on both software defined networking mechanism as well as virtual network function 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. [9] or more recently by Zargar et al. [10]. 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.

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. [11] 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 efficiency, will facilitate the assessment of software based responses.

Our work has shown that some of the components in a counter attacks are easily delivered using Virtual Network Functions. 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 [12] and mobile environments [13] [14]. 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 [15](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 [16] show the applicability of this type approach for identifying attacks. Our proposal to use machine learning to assess efficiency and adopt the most effective set of countermeasures is therefore a novel and promising application of such techniques.

9. 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. Finally, our measurements show that detection and response time are dependent on the attacks characteristics and continue our discussion on how we use impact and efficiency to evaluate different solutions. We conclude that metrics for impact of the attack and efficiency 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 efficiency 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[17].

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

[1] 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, OFC The Optical Networking and Communication Conference 2017 - under submission .

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

[3] 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, 305–316, 2010.

[4] R. Koning, N. Buraglio, C. de Laat, 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) .

[5] J. D. Howard, T. A. Longstaff, A common language for computer security incidents, Sandia National Laboratories .

[6] P. Quinn, T. Nadeau, Service function chaining problem statement, IEEF RFC 7498 .

[7] 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), 483–488, doi:\bibinfo{doi}{10.1109/NETSOFT.2016.7502489}, 2016.

[8] 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, 97–113, 2012.

[9] T. Peng, C. Leckie, K. Ramamohanarao, Survey of Network-based Defense Mechanisms Countering the DoS and DDoS Problems, ACM Comput. Surv. 39 (1), ISSN 0360-0300, doi:\bibinfo{doi}{10.1145/1216370.1216373}, URL http://doi.acm.org/10.1145/1216370.1216373.

[10] S. T. Zargar, J. Joshi, D. Tipper, A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks, IEEE Communications Surveys Tutorials 15 (4) (2013) 2046–2069, ISSN 1553-877X, doi:\bibinfo{doi}{10.1109/SURV.2013.031413.00127}.

[11] 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 Communications Surveys Tutorials 18 (1) (2016) 602–622, ISSN 1553-877X, doi:\bibinfo{doi}{10.1109/COMST.2015.2487361}.

[12] 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), 1622–1628, doi:\bibinfo{doi}{10.1109/ICACCI.2014.6968453}, 2014.

[13] H. Hawilo, A. Shami, M. Mirahmadi, R. Asal, NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC), IEEE Network 28 (6) (2014) 18–26, ISSN 0890-8044, doi:\bibinfo{doi}{10.1109/MNET.2014.6963800}.

[14] M. Chen, Y. Qian, S. Mao, W. Tang, X. Yang, Software-Defined Mobile Networks Security, Mobile Networks and Applications 21 (5) (2016) 729–743, ISSN 1572-8153, doi:\bibinfo{doi}{10.1007/s11036-015-0665-5}, URL http://dx.doi.org/10.1007/s11036-015-0665-5.

[15] R. Sommer, V. Paxson, Outside the Closed World: On Using Machine Learning for Network Intrusion Detection, in: 2010 IEEE Symposium on Security and Privacy, ISSN 1081-6011, 305–316, doi:\bibinfo{doi}{10.1109/SP.2010.25}, 2010.

[16] S. Ranjan, J. Robinson, F. Chen, Machine learning based botnet detection using real-time connectivity graph based traffic features, URL https://www.google.com/patents/US8762298, uS Patent 8,762,298, 2014.

[17] 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 .