2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)

Approaches for collaborative security defences in multi network environments

Ralph Koning, Gleb Polevoy, Lydia Meijer, Cees de Laat, Paola Grosso
System and Network Engineering group (SNE) University of Amsterdam, The Netherlands
Email: r.koning@uva.nl, g.polevoy@uva.nl, l.l.meijer@tno.nl, delaat@uva.nl, p.grosso@uva.nl

***Abstract*—Resolving distributed attacks benefits from collaboration between networks. We present three approaches for the same multi-domain defensive action that can be applied in such an alliance: 1) Counteract Everywhere, 2) Minimise Countermeasures, and 3) Minimise Propagation. First, we provide a formula to compute efficiency of a defence; then we use this formula to compute the efficiency of the approaches under various circumstances. Finally, we discuss how task execution order and timing influence defence efficiency. Our results show that the Minimise Propagation approach is the most efficient method when defending against the chosen attack.**

***Keywords*—Security, Defence Approaches, Multi-Domain Defence, Collaborative Defence, Defence Algorithms, Computer Networks**

I. INTRODUCTION

Attacks on computer infrastructures remain a persistent problem on the internet. Resolving these attacks often benefit from the support and actions of parties other than the victim. Still, most organisations connected to the internet operate their network domain autonomously using their own policies, therefore they share little information with others, which also applies to the subject of security.

To facilitate cooperation, Deljoo et al. [1] introduce the concept of security alliances, a framework in which groups of organisations can help each other to defend against attacks on one of the members in the alliance.

Security alliances [2] provide certain benefits and services such as:

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

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

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

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

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

Besides the organisational framework there needs to be some technical infrastructure to support such a collaboration; in our case we use the infrastructure provided by the SARNET [3] (Secure Autonomous Response NETworks) project.

In [4, Section 3.2] we introduced a method to evaluate the efficiency of defences in single domain environments. In Sec. IV we generalise this formula to allow for multiple parameters that influence the efficiency. We use this generalised formula to compute the efficiency of our multi-domain approaches.

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

The paper is organised as follows: In Sec. II we explain the SARNET framework we use for this work. Then, we describe the three different defence approaches in Sec. III and explain the generalised efficiency formula in Sec. IV. In Sec. V we explain how the approaches are implemented in our SARNET implementation, VNET, that we use to run our experiments. We discuss the attack scenario in Sec. VI and in Sec. VII we show the scenarios and conditions we used for the experiments of which we will show the results in Sec. VIII. Finally, we discuss our findings in Sec. IX and conclude and elaborate on the future work in Sec. XI.

II. MULTI-DOMAIN SARNET

SARNET is a framework for detection and mitigation of attacks on computer infrastructures, i.e. computer systems that are interconnected using networks to provide a service [5]. The framework collects *metrics* from both the network, the systems connected to the network, applications running on top of systems and performance metrics from higher level systems such as the amount of products that are sold during a time interval. An *observable* adds

978-1-7281-1661-7/19/$31.00 ©2019 IEEE
DOI 10.1109/CSCloud/EdgeCom.2019.000-9

a condition to the metric that can be monitored to see if the metric deviates from, thresholds, ratios, level based on historical values, and past behaviour. When the deviation is too large, depending on the condition, the observable changes from a healthy' to an 'unhealthy' state. One or more observables map to a *classification*; based on the classification further analysis may happen including database look-ups to collect the necessary information in order to decide what defences to pick. When multiple defences are available for the situation the SARNET picks the *defence* with the highest efficiency ranking and executes it. A defence consists of multiple *tasks*. After executing defensive actions, SARNET evaluates the metrics again and recalculates the efficiency rankings. If the system performance is still affected significantly by the attack, the cycle repeats and another defence is tried until all possibilities are exhausted.

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

A. Defence orchestration

Even when collaborating within an alliance, a member might not choose to share sensitive data with others because of company policies, or because laws and other norms [6] prevent them from doing so. For these reasons, we used a decentralised approach for defence orchestration. In our architecture is no central authority with a full overview of the situation and the domains themselves are responsible for building their own view of the alliance and the networks surrounding the alliance. We default to limited information sharing between the parties; the amount of information sharing can of course be increased when the situation requires it. Another advantage of this decentralised approach is the increase in robustness against attacks since there is no single point that can be attacked to cripple defence orchestration.

B. Responsibility

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

C. Agent communication

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

- *Control messages* for setting up and maintaining communication to other domains.
- *Informational messages* for requesting information and responding to the requested information.
- *Action messages* for implementing a certain countermeasure to reduce the attack impact.
- *Subscriptions* for longer lasting information or actions such as intelligence feeds or automatic protection.

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

III. Inter-domain defence strategies

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

- Algorithm 1 - Counteract Everywhere
- Algorithm 2 - Minimize Countermeasures
- Algorithm 3 - Minimize Propagation

Algorithm 1 shows the most aggressive approach. The approach immediately implements a countermeasure in the victim's domain as well in all other collaborating domains, if there is a pattern match. This results in a small impact because the countermeasure is applied as soon as possible, but since the countermeasure is applied on all alliance members this operation is costly (see Equation (1)):

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

Algorithm 2 reduces cost by only implementing countermeasures at members that see the attack pattern coming in from non-members. Since the nodes at the edge of the alliance E are a subset of all nodes N implementation costs (see Equation (2)) are equal to or less as in 1:

$$
\begin{aligned}
cost &= \sum_{e=1}^{E} C_e + P_e \times t \\
\text{where } N &= \text{participating nodes} \\
E &\subseteq N \\
e &\in E \\
C_e &= \text{ fixed cost of edge node } n \\
P_e &= \text{ cost of edge node } n \text{ per time period} \\
t &= \text{amount of time periods elapsed}
\end{aligned}
\tag{2}
$$

Algorithm 1: Counteract everywhere: asks for countermeasure from every member that sees atack traffic

```
Input: pattern: attack pattern
       alliance: alliance members
       N : victim's neighbours
for node ∈ N do
    request node for its neighbours that produce
     pattern;
    implement countermeasure at node;
    for neighbour ∈ neighbours do
        if neighbour ∈ alliance ∧ neighbour ∉ N
         then
            add neighbours to N;
        end
    end
end
```

Algorithm 2: Minimise countermeasures: places countermeasures only at where the attack traffic enters the alliance.

```
Input: pattern: attack pattern
       alliance: alliance members
       N : victim's neighbours
for node ∈ N do
    request node for its neighbours that produce
     pattern;
    if ∃neighbour ∉ alliance then
        implement countermeasure at node;
    end
    for neighbour ∈ neighbours do
        if neighbour ∈ alliance ∧ neighbour ∉ N
         then
            add neighbours to N;
        end
    end
end
```

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

Algorithm 3 is another optimisation of Algorithm 1. This approach reduces cost by reducing propagation, relying on recovery detection. The approach directly applies the countermeasure at the neighbours but instead of going to their neighbours directly it first waits for a time period defined by *wait time*. Only if the attack is not resolved, the approach request the nodes' neighbours for assistance. Because the approach tries to minimise the amount of assistance, the implementation costs will be lower (worst case they are equal to Equation (1)). Yet, waiting while under attack until we reach *wait time* has a negative effect on the impact of the attack compared to Algorithm 1 that does not wait.

The defence time for Algorithm 3 can improve when the *wait time* becomes smaller, although when the chosen *wait time* is too small ie. before the system can detect recovery,

Algorithm 3: Minimise propagation: minimises propagation of the countermeasure by filtering close to the victim.

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

the approach will continue asking other nodes similarly to Algorithm 1 but with a time penalty. Ideally, *wait time* is tuned to the time it takes for the victim to reliably detect recovery. This implies that the faster the victim can reliably detect recovery the more efficient Algorithm 3 becomes.

IV. EFFICIENCY

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

For example, during or after an attack we observe the deviation of each relevant metric from its baseline prior the attack. We define *impact* as the accumulated deviation from the start of the defence until the time of evaluation t. The higher the impact is, the lower the efficiency should be. On the other hand, efficiency is also constrained by the available budget to implement a countermeasure. The less budget is spent, the higher the efficiency becomes.

We evaluate the performance of the defence approaches by generalising Formula 1 from [4, Section 3.2]. In this original formula, the efficiency decreased in both allowed parameters. Now, we expand those formulas to allow for any finite number of parameters. Given a strictly increasing function f, such that $f(0) = 0$, we assume that efficiency decreases in $f(x_j)$ and increases in $f(y_i)$.

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

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

The efficiency formula utilises the following symbols:

- β defines the recovery - no recovery division point.
- The efficiency decreasing factors are denoted as $y_1, \ldots, y_m$; m is the amount of decreasing factors.
- The factors that increase efficiency are denoted as $x_{m+1}, \ldots, x_{m+l}$; l is the amount of increasing factors.
- The importance α generalises to $\alpha_1, \alpha_2, \ldots, \alpha_{m+l-1}$. The parameters α_i fulfil that $\sum_{i=1}^{l+m-1} \alpha_i$ is between 0 and $1-\beta$. The last factor, α_{m+l}, then implicitly gets the importance of $1 - \beta - \sum \alpha_1 \ldots \alpha_{m+l-1}$.

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

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

$$E(\text{recovered or not}, y_1, \ldots, y_m, x_{m+1}, \ldots, x_{m+l}) = \begin{cases} \beta + \sum_{i=1}^{m} \alpha_i \frac{f(y_i)}{f(Y_i)} + \sum_{j=m+1}^{m+l-1} \alpha_j \frac{f(X_j)-f(x_j)}{f(X_j)} \\ +(1 - \beta - \sum_{k=1}^{m+l-1} \alpha_k) \frac{f(X_{m+l})-f(x_{m+l})}{f(X_{m+l})} & \text{Recovered,} \\ \sum_{i=1}^{m} \alpha_i (\frac{\beta}{1-\beta}) \frac{f(y_i)}{f(Y_i)} \\ + \sum_{j=m+1}^{m+l-1} \alpha_j (\frac{\beta}{1-\beta}) \frac{f(X_j)-f(x_j)}{f(X_j)} \\ +(1 - \beta - \sum_{k=1}^{m+l-1} \alpha_k)(\frac{\beta}{1-\beta}) \frac{f(X_{m+l})-f(x_{m+l})}{f(X_{m+l})} & \text{otherwise.} \end{cases} \quad (3)$$

The full characterisation of efficiency is provided in [7].

Equation (4) in Sec. VII shows how we use Formula 3 to compare the efficiency of the approaches in Sec. III.

V. Implementation

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

A. VNET

We use the VNET emulation environment to instantiate the topologies in Sec. VII. VNET instantiates the topology as a network slice [8] on the ExoGENI cloud platform [9]. Each virtual machine on the ExoGENI platform runs a single domain in our multi-domain setup. We use two VM types, XOSmall (1 core, 1G RAM) for customer and transit domains and XOLarge (2 cores, 6G RAM) for the NFV and service domains. ExoGENI assures that the requested link capacity between the virtual machines is guaranteed by preventing overprovisioning and limits the bandwidth to the requested 100 megabit per link. The various components inside the domain are separated in Linux containers [10] using Docker [1] and communicate to each other using MQTT [2]. Routing between the domains is done with the Quagga software router [11] using the BGPv4 routing protocol [12]. Each domain runs an agent that talks to other domain agents and to our controller to configure and execute the attack scenarios. Defences are started autonomously when an attack is detected by the local SARNET agent that runs in each domain.

[1] https://docker.io

[2] http://mqtt.org

B. Topology building blocks

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

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

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

C. Inter-domain signalling protocol

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

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

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

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

Control messages:

- **Identify**: share identity and neighbours (for topology building)

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

Informational messages:

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

Action messages:

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

D. Algorithms implementations

The SARNET multi-domain agent allows domains to share intelligence and to collaborate with each other on both resolving and gathering information about attacks. Using flow information, the origin of an attack can be traced within a domain. Because aiding domains report back the neighbour from which the attack is coming, we can trace back attacks throughout the alliance to its border. Since the domains return where the traffic is really coming from and not where the traffic is supposed to be coming from, according to routing protocols, the defences work correctly even when traffic is spoofed.

Since we use a stateless approach in our network, the information requests and action tasks will be sent periodically for as long as the attack persists, both to continue monitoring whether the attack is still ongoing and to update defences as the attack characteristics change.

VI. Attack scenario

A Distributed Denial of Service (DDoS) attack is a good example of an attack that requires a collaborative response. Zagar et al. stipulate in their conclusion that collaboration, cooperation and distributed defences are key in defending DDoS attacks [13]. Most of the time the DDoS attack causes congestion on the link from an Internet Service Provider (ISP) to the victim. For mitigation to have effect, the victim has to ask their ISP to take action. Depending on the scale of the attack the bottleneck can also exist beyond the ISPs control, in which case other parties need to be included in the resolution of the problem.

We simulate this attack within the VNET topology by sending many small UDP packets at a chosen rate originating from the client domains. The amount of networks from where the DDoS can originate is constrained by the amount of client domains we requested in our topology, which is limited by the resources available on the ExoGENI rack. Therefore, the attack does not contain the amount of nodes and the bandwidth that are typical for a large DDoS attack. Still, the attack has multiple origins that need to be blocked in order for defence to have effect so the methods in this paper still apply when they are optimised for scale.

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

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

VII. Evaluation

We evaluate the strategies mentioned in Sec. III using the generic efficiency formula (see Sec. IV)).

The parameter β defines the cutoff point between recovered and not recovered. To compute efficiency, we chose a β of 0.2 to allow the efficiency in the *recovered* situation to be in the range from 0.2–1.0.

Efficiency uses multiple α to weigh the parameters by their importance of cost adding up to $1 - \beta$.

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

These weights do not include *cost*. We decided for an equal balance between the impacts of $rxBandwidth$ and *sales* as it comes to *cost*; so we will multiply each weight, $W_{rxBandwidth}$ and W_{sales}, by 0.5 such that *cost* can use the remaining 0.5.

Finally, we multiply the weights with $1 - \beta$, to ensure that the sum of each weight including costs equals $1 - \beta$ and that the weights can be used as α in the efficiency (Formula 3).

In a recovered scenario, Formula 3 reduces to the

following for the attack scenario we chose:

$$
\begin{aligned}
e_{rec} &= \beta + \alpha_S \frac{S-s}{S} + \alpha_T \frac{T-t}{T} + (1-\beta-\alpha_B)\frac{B-b}{B} \\
\text{where } S &= \text{max sales impact (no sales)} \\
s &= \text{actual sales impact } \textit{sales} \\
\alpha_S &= \text{importance of sales} \\
&= W_{sales} \times 0.5 \times (1-\beta) \\
T &= \text{max incoming traffic, link capacity} \\
t &= \text{actual incoming traffic, } rxBandwith \\
\alpha_T &= \text{importance of traffic} \\
&= W_{rxBandwidth} \times 0.5 \times (1-\beta) \\
B &= \text{budget} \\
b &= \text{budget spent, or } \textit{cost} \\
\alpha_B &= \text{importance of budget} \\
&= 1-\beta-(\alpha_S+\alpha_T) \\
e_{rec} &\rightarrow [\beta, 1]
\end{aligned}
\tag{4}
$$

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

A. Evaluation conditions

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

1) topology and alliance size: We run our experiments on two topologies:

1) a line topology, the alliance members are connected in a line. Each member can be connected to at max 2 other members. This is shown in Figure 1;
2) a tree topology, the alliance network forms a tree. Each member can be connected to 1 or 3 other members. This is shown in Figure 2.

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

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

Alliance size adjusts the depth of cooperating domains from *S1*. When size equals 0, there's no cooperation and *S1* acts on its own. When size equals 1, *51* can cooperate in attack mitigation; 2 includes *52* etc. By default the alliance size includes all transit domains and is set to 8. Note that the amount of members in the alliance is 9 because of the friendly *NFV domain*, *N1* connected to *51*, that is included when alliance size bigger than 1. In topology 2 all transit

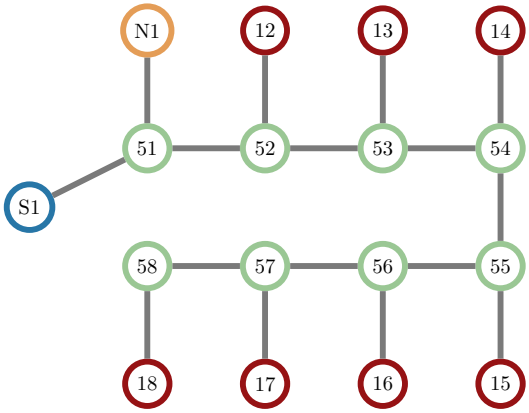


Figure 1: line topology

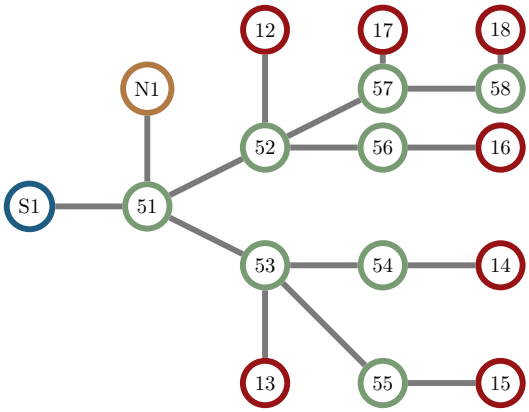


Figure 2: tree topology

nodes are already included at size 4, after which increasing the size has no effect anymore on the defence efficiency.

2) Costs and budget: Costs can differ per domain and consists of a fixed and a periodic part; the total costs are bound by the budget. For our experiments, we used a value of 0 credits for periodic cost and for the fixed cost we used a value of 100 credits. Practically, this means that every defensive action executed costs 100 credits and that we can limit the maximum budget to 900 credits (100 × alliance size). By default the budget is 900, which is enough credits to use the full alliance for the defence. Additionally, we run experiments for a budget of 300, and 600.

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

VIII. Results

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

- *Alliance size* (default=8), the distance from the victim that determines the domains inside the alliance (Figure 3);
- *Budget* (default=900), the amount of 'credits' that each domain has assigned for the task (Figure 4);
- *Attack size* (default=0.9), the accumulated strength of the attack (Figure 5).

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

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

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

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

A. Topology 1 - line

Figure 3a shows that in single attacker cases Algorithm 2 is performing best; for multi attacker cases Algorithm 3 performs better. In all the cases where one of the attackers is located far away Algorithm 1 performs poorly: for large alliances countermeasures are placed at all the nodes on the path, resulting in higher costs and lower efficiency.

Figure 4a shows a similar picture, yet Algorithm 1 is even performing worse under low-budget conditions. Still, Algorithms 2 and 3 are the best performers where Algorithm 2 is slightly better in single attacker scenarios.

When focusing on the effect of the attack size (see Figure 5a) we observe patterns similar to Figures 3a and 4a: Algorithms 2 and 3 perform better than Algorithm 1. However, Algorithm 3 seems to perform better than Algorithm 2 for small attack volumes. Similarly, Algorithm 2 is performing better than Algorithm 3 for larger attack sizes when there's *1 attacker far and 1 attacker close*.

B. Topology 2 -tree

When considering the alliance size in a tree topology we observe that Algorithm 3 performs better in all cases (See Figure 3b). In the *all clients attacking* scenario Algorithms 1 and 2 decrease more drastically; this is due to the tree topology that doubles the amount of attackers when the alliance size increases up until $alliancesize = 3$.

Figure 4b shows, similar to Figure 4a, that Algorithm 1 is the least efficient approach. In the *all clients attacking* scenario Algorithm 2 is performing comparably to Algorithm 1 and is not very efficient. This is due to the fact that also Algorithm 1 needs to implement the countermeasure at every node in the topology.

Figure 3b compared to Figure 3a shows less aggressive decline for Algorithm 3 in the *all clients attacking* scenario. Again Algorithm 3 is the most efficient except when the attack size is 2.0 in the *1 far 1 close* scenario where Algorithm 2 performs slightly better.

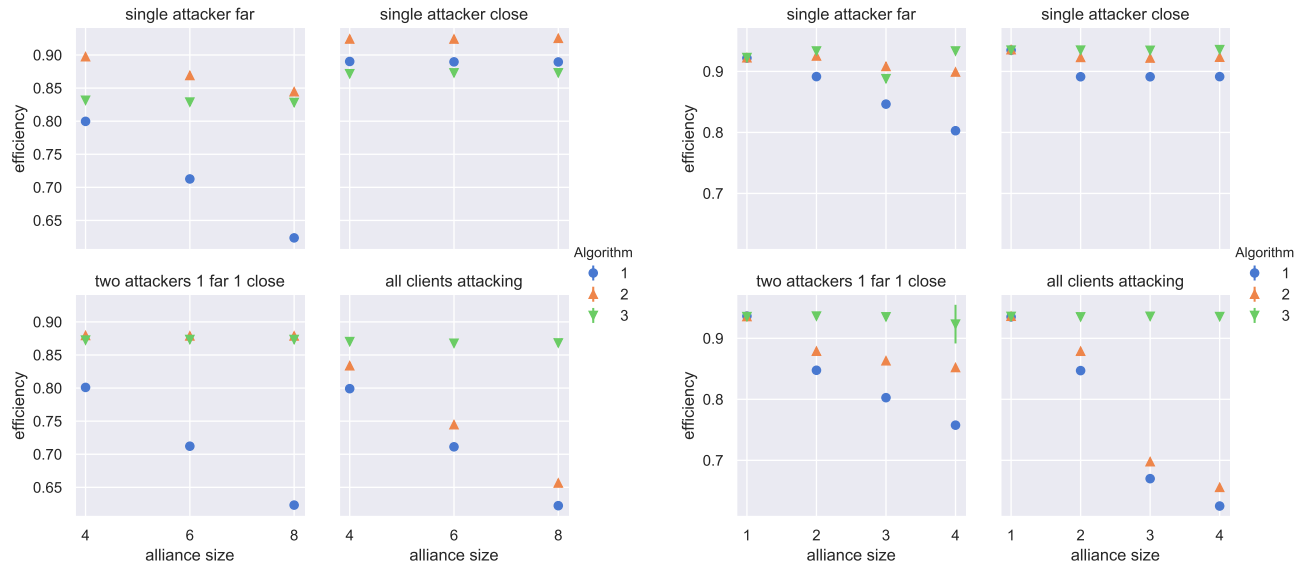
IX. Discussion

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

- Block everywhere: Algorithm 1 has the lowest efficiency because it is very costly.
- Minimised countermeasures: Algorithm 2 performs well in single attacker scenarios for the *line* topology. Since this is not the case for the *tree* topology and the efficiency is very close to Algorithm 3 it may not be worth it to make the exception unless very tight on budget.
- Minimised propagation: Algorithm 3 is in most cases the most efficient approach. The disadvantage is that the approach does not expel the attack from the alliance but just constrains the attack impact sufficiently from the victims' perspective. It is in fact the case that the attack traffic still passes through the alliance wasting resources from other members.

There are certain elements that affect the efficiency. First, consider the parameters in the efficiency formula. How α is chosen in the efficiency formula determines how much emphasis we put on certain factors; changing α will change the efficiency. In this case, since we only compare successfully defended scenarios; β acts as the limit on the range of efficiency.

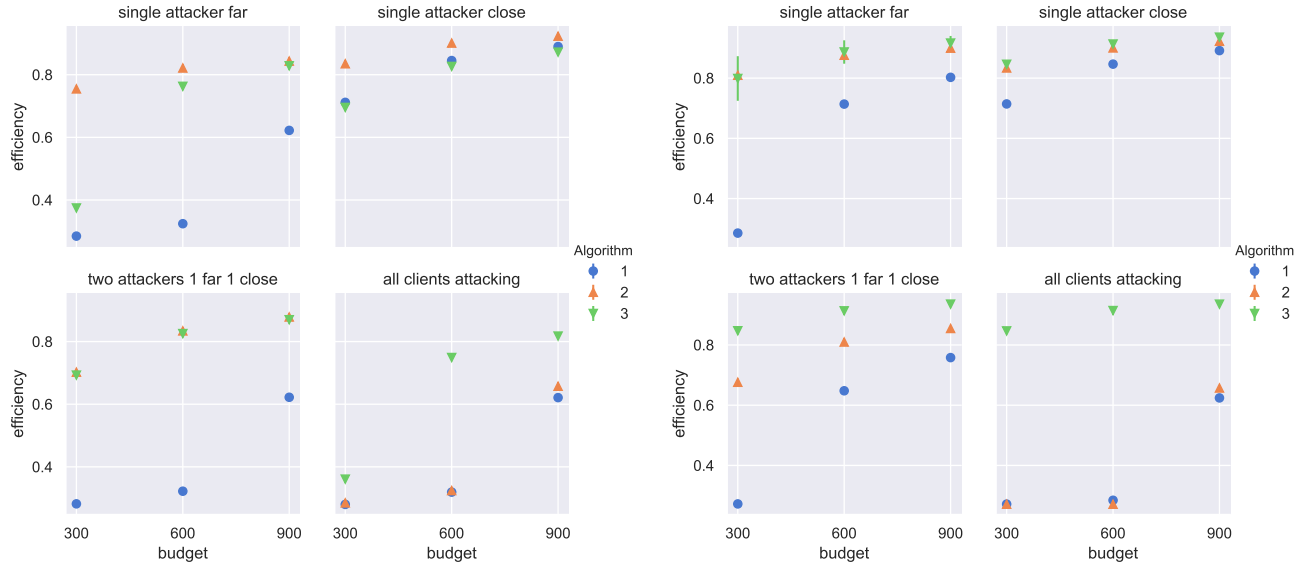
Second, the conditions under which the attack occurs influence the algorithms performance. Algorithms 1 and 2 are directly affected by the alliance size (Figure 3): in Algorithm 1 this has direct influence on the cost since countermeasures are applied at all nodes; in Algorithm 2 the alliance size has influence on the total time it takes to get the required information for every member before it can apply the countermeasure. Algorithm 3 is not directly influenced since the approach stops when the attack is mitigated.



(a) Line

(b) Tree

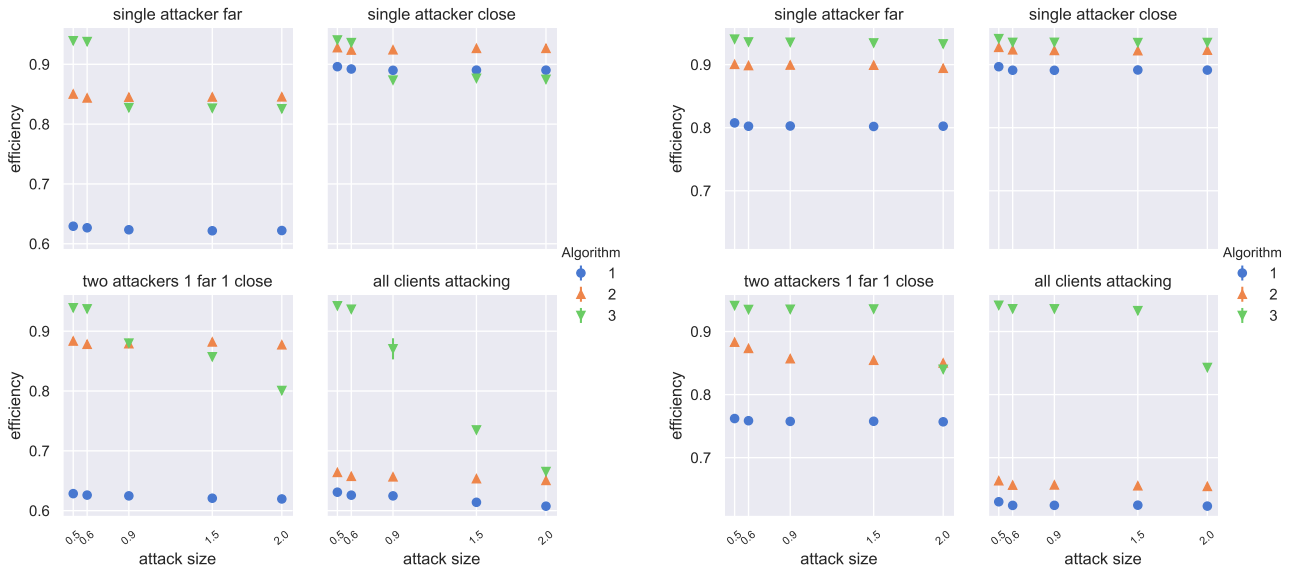
Figure 3: Alliance size



(a) Line

(b) Tree

Figure 4: Budget



(a) Line

(b) Tree

Figure 5: Attack size

Third, timings influence the performance of the algorithms; especially the time it takes to request information can negatively effect performance. In Algorithm 2 the algorithm first collects information for all nodes, if one of the nodes takes a long time to respond the countermeasure delays and efficiency goes down. The response time affects Algorithms 1 and 3 to a lesser degree, since they apply countermeasures on a per node basis. As mentioned in Sec. III Algorithm 3 is dependent on the *wait time* which is based on the time it takes to detect recovery. In our case this is set to the amount of seconds it takes to collect enough samples of our metrics to detect recovery: 16s, plus a 3s margin for the state to propagate through the system. Increasing this timing will increase the impact of the attack and make Algorithm 3 less efficient.

Finally, all algorithms are affected by budget constraints as can be seen in (Figure 4).

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

When starting our experiments, we expected that the *Minimise Countermeasure* approach (Algorithm 2) would be the most efficient under budget constraints. This approach blocks the attacks at the source and is considered to be an effective approach against DDoS attacks [14, 15]. Instead, we found that the *Minimise Propagation* approach (Algorithm 3) was more efficient. *Minimise Propagation* mitigates attacks sooner because this approach does not have to trace the attack back to the attackers; the approach also works with less countermeasures placed because it stops when the attack impact is sufficiently reduced. Both factors influence efficiency positively.

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

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

X. Related Work

A significant amount of research can be found on collaborative DDoS detection and response. Most of the papers focus on multi-domain detection [16]–[19], multi domain defence [20, 21] and combinations of the two [22]. All of these articles focus on a single attack and DDoS specificly. We use DDoS response only as a basic use case, yet the approaches that we evaluate in this paper can be used for asking assistance from friendly domains for *any* attack pattern. The collaborative defence approaches in this paper can be used as a base for defending against other attacks.

Our work presumes that all participating domains in an alliance are willing to cooperate. In principle one would want to verify that the other parties are really acting upon each request. Mannhart et al. [23] discusses four approaches to ensure effectiveness of a cooperative mitigation. The work focuses on validating that the 'mitigator' correctly applied the mitigation. The authors pursue tamper-proof execution and verification of an applied countermeasure. They conclude that none of their four approaches alone is capable of providing this. Since we expect that the collaboration happens in an alliance context, there is a basic level of trust present between the members including rules and regulations on how to handle other members' data. The evaluation of the defence happens from the victims perspective and does not rely on any guarantees from its collaborators.

Meng et al. wrote a taxonomy on collaborative security systems [24]. They assess multiple security related systems that use both collaborative detection and collaborative response to achieve their goals. They identify a number of challenges in this type of collaborative systems. Our work addresses a number of them: in regard to privacy we adopt a limited information sharing mechanism (see Sec. II-A); incentive is in our case implicit in the existence of the security alliance; when looking at robustness we avoid a single point of failure by not relying on a central coordinator.

XI. Conclusion

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

Furthermore, our results showed that *Minimise Propagation* (Algorithm 3) is the most efficient approach to take when defending against a DDoS attack. System timings are another important factor: if tasks are dependent on each other and the first task takes a long time to complete, the second task has to wait. This increases the overall run-time of the defence, which impacts its efficiency. Waiting for the system to determine recovery also negatively impacts the defence efficiency.

Our future work will focus on evaluating a fourth approach that uses the Social Computational Trust Model described in [25]. We believe that alliance members would rather ask for support from members that showed that they are capable and willing to help (*evidence-based trust*). We want to assess if such an approach based on these trust values shows better efficiency than the ones evaluated here.

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

Finally, it is interesting to research how dynamic costs affect efficiency. In this paper we did not use periodic or dynamic costs; we also kept the costs equal at all collaborators. Introducing dynamic budgets based on attack traffic or on periodic cost, for as long as the countermeasure is active, has an effect on efficiency. Due to the dynamic cost, defence costs are not known a-priory and since active defences consume budget over time one may have to switch to another defence approach halfway in order to remain efficient.

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). We would like to thank Ben de Graaff, for his contributions to this project. Special thanks to Ciena for hosting our demonstration at their booth at SC17 and in particular Rodney Wilson, Marc Lyonnais, and Gauravdeep Shami for providing feedback on our work and their continuous support. We would also like to thank our other research partners TNO and KLM.

References

[1] 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 Soft ware-defined Exchanges*, 2016.

[2] A. Deljoo, T. van Engers, R. Koning, L. Gommans, and C. de Laat, "Towards trustworthy information sharing by creating cyber security alliances," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, Aug 2018, pp. 1506–1510.

[3] R. Koning, A. Deljoo, S. Trajanovski, de Graaff, Ben, P. Grosso, L. Gommans, T. van Engers, F. Fransen, R. Meijer, R. Wilson, and C. de Laat, "Enabling e-science applications with dynamic optical networks: Secure autonomous response networks," *OFC The Optical Networking and Communication Conference 2017*, 2017.

[4] R. Koning, B. de Graaff, G. Polevoy, R. Meijer, C. de Laat, and P. Grosso, "Measuring the efficiency of sdn mitigations against attacks on computer infrastructures," *Future Generation Computer Systems*, vol. 91, pp. 144–156, 2019.

[5] R. Koning, P. Grosso, and C. de Laat, "Using ontologies for resource description in the cinegrid exchange," *Future Generation Computer Systems*, vol. 27, no. 7, pp. 960–965, 2011.

[6] D. Basin, S. Debois, and T. Hildebrandt, "On purpose and by necessity: compliance under the gdpr," *Proceedings of Financial Cryptography and Data Security*, vol. 18, 2018.

[7] G. Polevoy, "Defence efficiency," 2019, arXiv:1904.07141 [cs.PF].

[8] Y. Yao, Q. Cao, J. Chase, P. Ruth, I. Baldin, Y. Xin, and A. Mandal, "Slice-based network transit service: Inter-domain l2 networking on exogeni," in *Computer Communications Workshops (INFOCOM WKSHPS), 2017 IEEE Conference on*. IEEE, 2017, pp. 736–741.

[9] I. Baldin, J. Chase, Y. Xin, A. Mandal, P. Ruth, C. Castillo, V. Orlikowski, C. Heermann, and J. Mills, "Exogeni: A multi-domain infrastructure-as-a-service testbed," in *The GENI Book*. Springer, 2016, pp. 279–315.

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

[11] P. Jakma and D. Lamparter, "Introduction to the quagga routing suite," *IEEE Network*, vol. 28, no. 2, pp. 42–48, 2014.

[12] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (bgp-4)," ietf, Tech. Rep., 2005.

[13] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," *IEEE communications surveys & tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[14] J. Mirkovic, G. Prier, and P. Reiher, "Attacking DDoS at the source," in *10th IEEE International Conference on Network Protocols, 2002. Proceedings.*, Nov. 2002, pp. 312–321.

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

[16] Y. Chen, K. Hwang, and W. Ku., "Collaborative detection of ddos attacks over multiple network domains," *IEEE Transactions on Parallel & Distributed Systems*, vol. 18, pp. 1649–1662, 06 2007. [Online]. Available: doi.ieeecomputersociety.org/10.1109/TPDS.2007.1111

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

[18] L. Zhu, X. Tang, M. Shen, X. Du, and M. Guizani, "Privacy-preserving ddos attack detection using cross-domain traffic in software defined networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 628–643, 2018.

[19] S. Suresh and N. S. Ram, "Feasible ddos attack source traceback scheme by deterministic multiple packet marking mechanism," *The Journal of Supercomputing*, pp. 1–15, 2018.

[20] J. Steinberger, B. Kuhnert, A. Sperotto, H. Baier, and A. Pras, "Collaborative ddos defense using flow-based security event information." in *NOMS*, vol. 2016, 2016, pp. 516–522.

[21] J. Steinberger, B. Kuhnert, C. Dietz, L. Ball, A. Sperotto, H. Baier, A. Pras, and G. Dreo, "Ddos defense using mtd and sdn," in *16th IEEE/IFIP Network Operations and Management Symposium 2018: Cognitive Management in a Cyber World*, 2018.

[22] R. Kesavamoorthy and K. R. Soundar, "Swarm intelligence based autonomous ddos attack detection and defense using multi agent system," *Cluster Computing*, pp. 1–8, 2018.

[23] S. Mannhart, B. Rodrigues, E. Scheid, S. S. Kanhere, and B. Stiller, "Toward mitigation-as-a-service in cooperative network defenses," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2018, pp. 362–367.

[24] G. Meng, Y. Liu, J. Zhang, A. Pokluda, and R. Boutaba, "Collaborative security: A survey and taxonomy," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 1:1–1:42, Jul. 2015. [Online]. Available: http://doi.acm.org/10.1145/2785733

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