SDN based scalable MTD solution in Cloud Network

Ankur Chowdhary Sandeep Pisharody Dijiang Huang
School of Computing, Informatics and Decision Systems Engineering
Arizona State University, Tempe, AZ
<achaud16, spishar1, dhuang8>@asu.edu

ABSTRACT

Software-Defined Networking (SDN) has emerged as a framework for centralized command and control in cloud data centric environments. SDN separates data and control plane, which provides network administrator better visibility and policy enforcement capability compared to traditional networks. The SDN controller can assess reachability information of all the hosts in a network. There are many critical assets in a network which can be compromised by a malicious attacker through a multistage attack. Thus we make use of centralized controller to assess the security state of the entire network and pro-actively perform attack analysis and countermeasure selection. This approach is also known as Moving Target Defense (MTD). We use the SDN controller to assess the attack scenarios through scalable Attack Graphs (AG) and select necessary countermeasures to perform network reconfiguration to counter network attacks. Moreover, our framework has a comprehensive conflict detection and resolution module that ensures that no two flow rules in a distributed SDN-based cloud environment have conflicts at any layer; thereby assuring consistent conflict-free policy implementation and preventing information leakage.

Keywords

Software-Defined Networking (SDN), Moving Target Defense (MTD), Attack Graph (AG)

1. INTRODUCTION

Moving Target Defense (MTD) [13, 25] is a transformative approach to security of multi-tenant cloud environment that leverages the dynamism in computer systems to create an environment that has a changing attack surface. This dynamism gives rise to the need for a framework to accurately, and in a timely fashion, examine the complex relationships between various hosts and security vulnerabilities in an ever changing cloud networking environment, and ensure that any changes made to the environment do not conflict with security policies.

MTD'16, October 24 2016, Vienna, Austria

DOI: http://dx.doi.org/10.1145/2995272.2995274

Traditional networks are composed of heterogeneous elements such as routers, firewall, switches. Each of these devices has their own proprietary software and protocols. This kind of network setup leaves very little scope for innovation in network. Software-Defined Networking (SDN) has emerged as a solution for addressing this challenge. SDN environment generally consists of OpenFlow switches and controllers, communicating over a secure channel. SDN provides a service oriented architecture to deploy modular solutions for different requirements.

The SDN controller can work as a centralized security policy enforcer. Ethane [6] provides a model for user authentication and stronger binding between packets and origin.

The programmable interfaces afforded by SDN can be conformed to achieve a dynamic defensive strategy based MTD [23]; thereby providing a systematic solution by selecting countermeasures to prevent or mitigate attacks in an SDN enabled data center networking environment [8, 15]. SDN based random host mutation [15] makes use of mapping between real and virtual IP addresses to make a reconnaissance difficult for attackers with the SDN controller providing centralized management of mutation control across the network.

In an Infrastructure-as-a-Service (IaaS) cloud, Virtual Machines (VMs) are managed by tenants and may contain various vulnerabilities and thus they are easy targets for attackers. Chung et al. [7] present an MTD approach to automate an iterative three-step procedure to counter network attacks, *a*) network intrusion detection; *b*) threat analysis; and *c*) countermeasure selection and deployment. To this end, we use an Attack Graph (AG) based vulnerability analysis model to enumerate all possible attack scenarios, allowing the cloud system to select countermeasures before identified vulnerabilities are exploited. Selected countermeasures could be based on link and network layer operations and reconfiguration, such as *a*) traffic redirection; *b*) traffic blocking/dropping; *c*) quarantining; *d*) MAC/IP addresses reconfiguration; etc. Since changes for the network are applied in an SDN based IaaS cloud by generating new flow rules that are added to the controller, it is necessary to check and resolve conflicts with existing rules, enterprise security policies and business Service Level Agreements (SLA) before such policies are deployed. The policy conflict problem becomes even harder in SDN enabled cloud networking setup since each SDN switch, both physical and software switches, can be considered a distributed firewall instance. Additionally, SDN based policy checking demands a cross-layer checking since flow rules may be defined at different protocol layers [20].

To address the described issues, in this research, we present a novel framework that includes a stable converged AG even in a dynamic environment. We incorporate the programmable network capabilities of SDN to isolate, quarantine, and inspect traffic and leverage these capabilities to implement MTD countermeasure strategies. Finally, we run candidate countermeasures through a policy conflict resolution module that ensures it does not conflict with existing security policy or business SLAs. The key contribution of our work is:

- We automate dynamic system reconfiguration safely by leveraging scalable AG and cross-layer security policy checking. It requires no involvement by network operators.
- We are able to successfully implement a framework that does real-time network reconfiguration either proactively, or reactively to any abnormal events in the environment.
- The reconfigured system is guaranteed to be compliant with security and SLA requirements of the organization.

This paper is organized as follows. In Section 2, we present some background and a motivation scenario for the presented research. The system models and architecture are presented in Section 3. The detailed description of our framework is presented in Section 4. The performance evaluation is presented in Section 5 and related work is presented in Section 6. Finally, we conclude our work and future research directions in Section 7.

2. BACKGROUND

Attack graphs (AG) are a good tool to represent the security state of entire network. AGs have proved to be a very useful tool to detect multi-staged and multi-hop attacks which may not be obvious to the network administrator by plain analysis. Some of the earlier works in this field have used model checking [27, 28] and formal language based methods [28] to enumerate all possible attack scenarios in the cloud system. We use AG based analysis in our threat model for our work.

Moving Target Defense (MTD) techniques have been devised as a tactic wherein security of a system is enhanced by having a rapidly evolving system with a variable attack surface; thereby giving defenders an inherent information advantage. An effective countermeasure used in MTD is network address switching, which can be accomplished in SDN with great ease.

In the data center network shown in Figure 1, we have Tenant A hosting a web farm. Being security conscious, only traffic on TCP port 443 is allowed into the IP addresses that belong to the web servers. When an attack directed against host $A2$ has been detected, the MTD application responds with countermeasures and takes two actions: *a*) a new webserver (host $A3$) is spawned to handle the load of host $A2$; and *b*) the IP for host $A2$ is migrated to the Honeypot network and assigned to host Z.

In order to run forensics, isolate and incapacitate the attacker, the HoneyPot network permits all inbound traffic, but no traffic outward to other sections of the data center. These actions result in new flow rules being injected into the flow table that *a*) permits *all* traffic inbound to the IP that originally belonged to host $A2$, but now belongs to host Z;

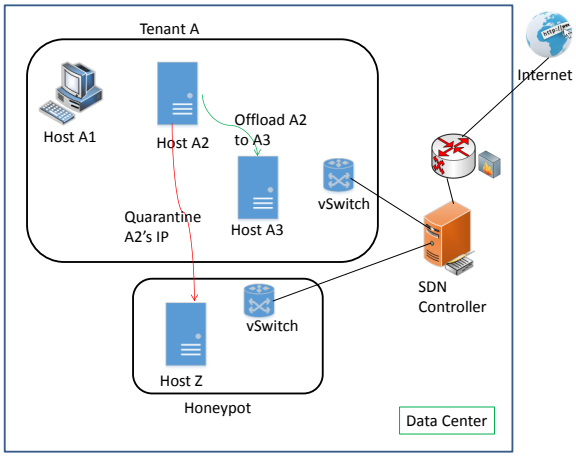

Figure 1: Motivating scenario for policy conflict.

b) modifies an incoming packet's destination address from host $A2$ to host $A3$ if the source is considered to be a non-adversarial source; *c*) stops all outbound traffic from the IP that originally belonged to host $A2$, but now belongs to host Z to the rest of the data center; and *d*) permits traffic on port 443 to host $A3$ (not of great importance to our case). The original policy allowing only port 443 to the IP of host $A2$, and the new policy allowing all traffic to the IP address of host Z are now in conflict.

3. SYSTEM ARCHITECTURE & MODEL

In this section, we describe system architecture of our SDN based MTD system, show how to use a distributed Scenario Attack Graph (SAG) model to describe the possible threats and vulnerabilities in a large network, and propose a policy conflict checking model to address the possible conflicts from mitigation strategies of the countermeasure selection.

3.1 Threat Model

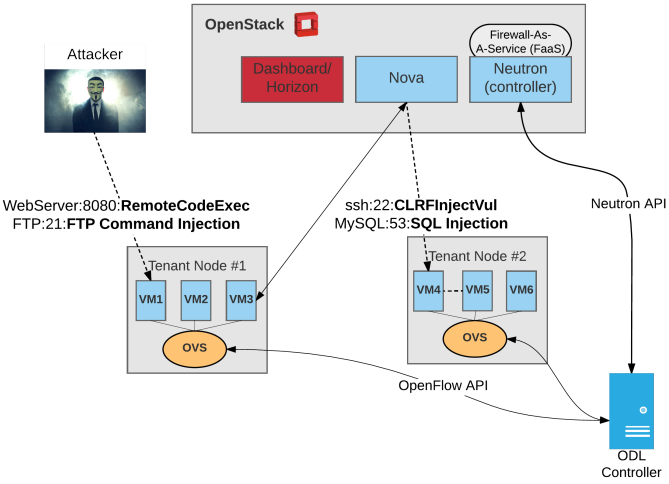

Figure 2: Threat Model.

In our attack model, we assume that an attacker can be located either outside or inside of the virtual networking system. The attackers primary goal is to exploit vulnerable VMs and compromise them as zombies. Our protection model focuses on virtual network based attack detection and reconfiguration solutions to improve the resiliency to zombie explorations. Our work does not involve a host based IDS and does not address how to handle encrypted traffic for attack detections. Our proposed solution can be deployed in an Infrastructure-as-a-Service (IaaS) cloud networking system, where the Cloud Service Provider (CSP) is benign. We also assume that cloud service users are free to install any operating systems or applications, even if they are known to be from adversarial sources.

We consider an OpenStack based cloud networking environment Figure 2. The components in the figure are OpenStack modules responsible for various functions. Nova is responsible for VM provisioning and management. Neutron provides network control. We install Firewall-as-a-Service (FaaS) on top of Neutron. Neutron interacts with OpenDaylight (ODL) controller through a REST API. Various firewall operations at layer-2 or layer-3 in this overlay network can be deployed to VM's ODL policies via Neutron.

If we assume one VM of tenant 1 has a web server running and another VM on tenant 2 has database server running. If the goal of attacking is to compromise database server, the attacker can first compromise web server on tenant 1 via remote code execution or FTP based vulnerability. Once he/she has access to the web server, he/she can use a web server as a zombie VM and compromise the database server on tenant 2 using SQL injection or SSH CLRF injection vulnerabilities.

3.2 System Architecture

Our system provides a framework to implement effective MTD solutions while ensuring that the implemented solutions stay consistent with the overall security policy, and do not cause unexpected side-effects as a result of conflicts with other flow rules present in the system. This framework utilizes the current system configuration, security policies, flow rules, and a vulnerability database as input. It follows a Detect-Analyze-Counter (DAC) cycle to make real time updates to the network environment and effectively track any system state or policy changes. This is accomplished through four functionally distinct modules, as shown in Figure 3: *a*) attack and information gathering module; *b*) attack analytics module; *c*) countermeasure selection and enforcement module; and *d*) policy conflict checking and resolution module. The attack and information gathering component consists of a vulnerability information gathering module. This information is sent to the attack analytics module which conducts AG processing and analysis to determine the current exposure to vulnerabilities. Based on the node vulnerability and reachability information and flow rules obtained from the SDN environment, this module performs node based security classification and stores the information in a database with the node acting as a primary key. This information is then passed to countermeasure selection and enforcement module. A candidate countermeasure is selected, and evaluated for consistency with the organization's security policy and SLAs. Upon acceptance, these new flow and connectivity information is then fed back to the attack and information gathering module. This creates a dynamic

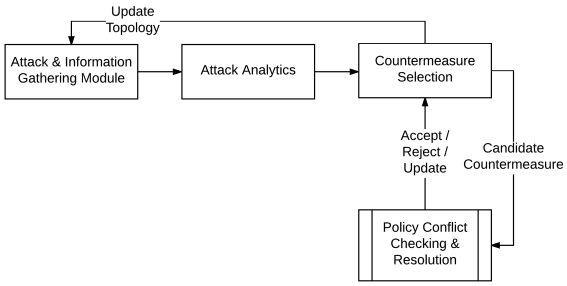

Figure 3: System Architecture.

DAC cycle, which ensures the system's security compliance and real time visualization.

Figure 4 represents the overlay structure of networks in an SDN environment, and the layer on which each of the modules operates. Note that the modules function independent of the physical topology, as is typical in any cloud environment. The SAG generation and countermeasure strategy is based on the overlay network topology. The candidate countermeasures are suggested as modifications to the overlay network.

3.3 Attack Analysis Model

Definition 1. *A hypergraph is a generalization of a normal graph* $H = (V, E)$. *V represents vertices* $V = \{v_i | 1 \leq i \leq n\}$ *and* $E = \{e_j | 1 \leq j \leq m\}$ *represent the hyperedges. A hyperedge is a subset of vertices. The degree of vertex is the number of hyperedges it is part of. Degree of a given vertex* $d(v_i) = |e_j| \geq 2$.

Definition 2. *A node* $N = \{H, V\}$ *in an AG is a combination of hosts in the environment and the possible vulnerabilities that exist on that particular host.*

Definition 3. *An Attack Graph (AG) is a tuple* $G = \{S, \tau, s_0, s_t\}$ *where* $S = N$ *is number of states possible,* $\tau \subseteq S*S$. $s_0 \subseteq S$ *is set of initial states and* $s_t \subseteq S$ *is set of success states.*

Definition 4. *Hypergraph partitioning is the process of partitioning a hypergraph into k-way* ($k \geq 2$) *disjoint set of node blocks* B_1, B_2, B_k *such that* $B_i \cap B_j = \emptyset \ \forall i \neq j$. *Thus, we are trying to find a k-way mincut in a hypergraph.*

Definition 5. *A Scenario Attack Graph (SAG) = (V,E) where*

1. $V = N_C \cup N_D \cup N_R$ *denotes a set of vertices that includes conjunction nodes* (N_C) *to represent exploits, disjunction node* (N_D) *denoting results of exploit, and root node* N_R *denoting the initial step of an attack scenario.*
2. $E = E_{pre} \cup E_{post}$ *denotes the set of directed edges. An edge* $e \in E_{pre} \subseteq N_D * N_C$ *denotes that* N_C *can be attained only if* N_D *is satisfied. An edge* $e \in E_{post} \subseteq N_C * N_D$ *means that* N_D *can be obtained if* N_C *is satisfied.*

As can be seen from Figure 2 the threat model under consideration has four nodes. We denote the web server running on VM1 `WS`, the database server on VM4 as `DS` and the attacker as `A`. The following is a possible chaining of exploits by the attacker:

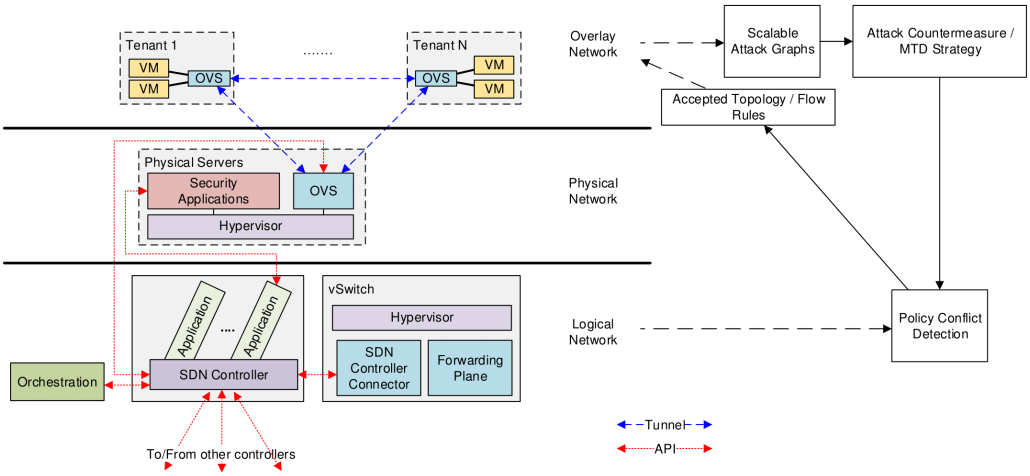

Figure 4: System modules and operating layers.

- `execCode(A,WS,8080)` - Remote Code exploit.
- `execCode(A,WS,21)` - FTP vulnerability exploit.
- `execCode(WS,DS,22)` - CLRF inject vulnerability exploit.
- `execCode(WS,DS,53)` - SQL injecttion exploit.

If there are N services each on the two hosts VM1 and VM2, a fully connected AG will have N^2 nodes. It is easy to see how the size of the nodes in AG can soon grow out of bounds, making it hard to comprehend and represent each possible attack scenario.

Our experimental results on Mininet [1] using an ODL controller shows that it takes over twenty minutes to generate an AG for a network of ten thousand nodes. We plan to use the SDN based ODL controller in cloud infrastructure to manage the complexity of assessing the security state of the entire network. A MTD solution for such a large network will face scalability challenges - state space explosion, something our MTD solution effectively addresses.

We partition the large AG into Sub Attack Graph (SubAG) using the SDN controller as driver program. The driver program coordinates with the sub attack graph creation modules known as SubAG Agents to construct a full AG. This helps in fast real time attack scenario analysis. The process of the hypergraph partitioning based AG creation reduces the time to construct full AG. The AG constructed helps in MTD countermeasure selection in real-time. For instance, in Figure 5 the hosts are partitioned into two SubAGs each shown with different color. Services SSH and FTP on hosts VM1, VM2, VM3; and ISCSI, SSH and MySQL on VM4, VM5, VM6 represent two partitions for the AG. The key idea is to distribute the load of SubAG creation over several processors for each tenant and then check reachability links across the tenants. The attacker in Figure 5 has root access on VM1, so he is able to exploit service MySQL based vulnerability on VM4. We merge reachability information across SubAG's to get final the AG. The number of processors depends upon Cluster on which Openstack is deployed. For instance, if we are using HP Blade Server Cluster with 10 CPUs allocated to the task of partitioning, with 2 processors per CPU, value of the number of processors is 20.

Attack graphs are also useful in identifying the critical

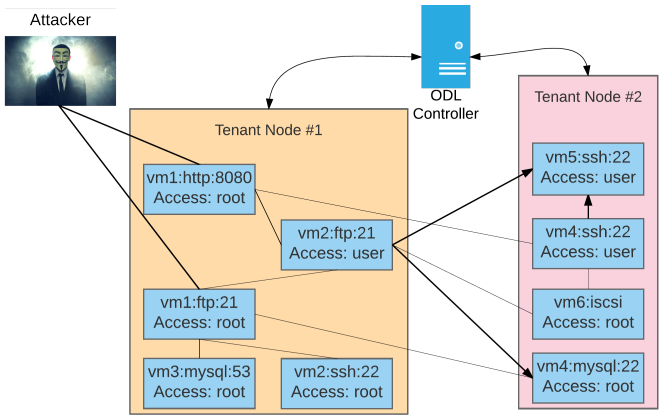

Figure 5: A network reachability based partitioned Attack Graph

assets in the network, for instance Asset Rank [24] based approach can be used to rank critical assets in network, which are more likely to be affected as a result of network vulnerabilities. We can prioritize these assets for MTD countermeasure.

The attack graph based approach is very effective in handling the dynamic attacks such as DDoS. If a botnet server communicates with clients to target a system resource, this information can be modeled using Attack Graph. Most modern vulnerability scanners report only the known vulnerabilities in the network such as *remoteCodeExecution, localBufferOverflow,* etc. An attacker, however on gaining privileges on a machine can install malicious applications that can trigger zero day vulnerabilities. For instance, in the Figure 5 VM1 which is WebServer may be providing some web service to VM2. Once an attacker has gained root access on VM1, he can downgrade a patched software, e.g- flash player to a vulnerable version and try to compromise

VM1. These unknown security risks can also be modeled using Attack Graphs. Although not discussed in the current paper, we plan on exploring this active area of research using attack graphs.

3.4 Policy Conflict Checking Model

Definition 6. *A countermeasure is a set of operations on a system that results in updating network configuration or traffic policy that thwarts attacks while adhering to system security policies and providing consistently high user experience.*

Implementing any countermeasure in an SDN environment would require a new set of flow rules to be generated. However, introducing new flow rules as part of the countermeasure selection brings to light complex issues, amongst which are cross-layer policy conflicts. Since abstracting the data plane from the control plane in SDN means that multiple users could share the same physical network; if a tenant were to implement an MTD solution, they could be implementing flow rules without holistic system knowledge. While these flow rules are implemented perceivably private security policies, their presence on a shared control plane means it could lead to potential flow rule conflicts in the overall environment.

In a flow table F containing rule set $\{r_1, r_2, ..., r_n\}$, with each flow rule $r_i \in R$ being a 6-tuple $< p_i, \alpha_i, \nu_i, \epsilon_i, \rho_i, a_i)$, we have *a*) p as the priority of the rule defined in the range $[1, 65535]$; *b*) α is the ingress port; *c*) ν is a VLAN ID; *d*) ϵ is a 6-tuple having source and destination addresses at OSI layer-2, layer-3 and layer-4; *e*) ρ as the layer-4 protocol; and *f*) a as the action set for the rule. Match fields α and ν, representing ingress port and VLAN ID merely eliminate and not add to potential conflicts. Hence, we do not include them in further discussion. Flow rules also contain packet counters and timeout values, but they are not relevant match or action fields in rule processing.

Since flow rules in an SDN environment are clearly a super-set of rules in a traditional firewall environment, work on flow rule conflicts are an extension of the work on firewall rule conflicts. While several works have classified firewall rule conflicts ;the seminal work of Al-Shaer and Hamed [4] is often used to classify firewall rule conflicts in a single firewall environment. Pisharody et al. [20] introduced a new classification of conflicts that better describes conflicts between address space overlap over multiple OSI layers.

Knowing that OpenFlow specifications clarify that if a packet matches two flow rules, only the flow rule with the highest priority is invoked, conflicts in SDN flow rules can be classified into categories *Redundancy, Generalization, Correlation, Shadowing, Overlap, Imbrication* as discussed by Pisharody et al. [20].

3.5 System Safety and Stability Post Countermeasures

As discussed during countermeasure selection algorithm 4 the VM is migrated from one physical server to another in way that the overall vulnerability score for system is decreased. We still need to ensure no new attack scenarios from reshuffling impact system safety. In addition, migration can impact the service availability for clients accessing these VMs. We use formal methods post-countermeasure to ensure *System Liveliness* and *System Safety* [5]. We identify all preconditions for liveliness as P_{live} and P_{safe} [5]. For example,

$$P_{safe} = \{\neg root(WS), \neg localprivEsc(ftpServ), ...\}$$

and

$$P_{live} = \{sshAccess(VM1, VM2), ftpAccess(VM1, ftpServ)\}$$

Starting state of system is $s_0 = \{userAccess(VM1), userAccess(WS), ..\}$ and the terminating state of the system is s_t.

For the attack graph G, we check preconditions nodes described in attack analysis model $e \in E_{pre} \subseteq N_D * N_C$, then $N_C \models P_{safe} \cup P_{live}$. We ensure all preconditions lead to a state s_t which belongs to a critical asset to satisfy safety and liveliness properties.

4. IMPLEMENTATION

In this section, we discuss the implementation details for modules that comprise our framework. First, we discuss algorithms for a scalable AG, followed by a countermeasure selection module and then the policy conflict detection and resolution module. We note that the present system assumes the intrusion detection already in place and it will send alerts to the security analysis module when malicious attacks are detected.

The overall architecture of AG generation involves the creation of a reachability and vulnerability based hypergraph on each tenant. The decision on creation of hypergraph is based on size of number of nodes per tenant. A generated hypergraph is partitioned based on number of processors allocated by the administrator. For example, if three processors are allocated for a given tenant, then each processor will take care of the creation of a SubAG, and we will have three SubAGs. We use parallelization based on PySpark [3] framework to merge the SubAG for each tenant. Resilient Distributed Datasets (RDDs) [33] are parallel data-structures that are used for information exchange in distributed networks. RDDs exchanges post conditions generated by each SubAG and attack tenant AG till no new post conditions are left.

4.1 Attack Analysis Algorithm

We improve scalability of the attack analysis algorithm by using parallelization framework and partitioning of large hypergraph [17] into the logical framework of SubAG. The data structures such as RDD are used for an efficient hypergraph partitioning.

Algorithm 1 is responsible for creating a hypergraph and

Algorithm 1 Hypergraph Partitioning Algorithm

```
1: procedure PartHypergraph(G, k ≥ 2, p, H ← ∅)
2:     V ← v_1, v_2, ...., v_n
3:     for i = 1 → (n) do
4:         H ← H ∪ {v_i, N(v_i)}
5:     edgecuts, parts ← ParMETIS(H, k)
6:     return SHG
```

partitioning the hypergraph based on parallel hypergraph partitioning algorithm ParMETIS [17]. The algorithm takes

graph connectivity information represented by G as input, along with a number of processors p, number of partitions required k and empty data structure for hypergraph H. The vertices in neighborhood of the vertex $N(v)$ are part of hyperedge.

The Hypergraph H is then partitioned into regions based on number of tenants as shown in use case Figure 5 using the partitioning algorithm. $SHG[n_i]$ will return partition to which node n_i belongs to after partitioning. For example, in Figure 5, `SHG[vm1:http:8080]` will return `Tenant Node 1`. Each processor p is responsible for construction of a SubAG or cluster for a given tenant. This distribution of load for AG construction is done using a PySpark [3] based framework which maps SubAG construction phase over the p processors.

Hypergraph partitioning process aims to find k-way min-cut in a Hypergraph, with k>1. Consider a variable ϵ such that $0 < \epsilon < 1$. The objective of partitioning algorithm is to construct a partition set $\pi = \{B_1, B_2, .., B_k\}$ from the hypergraph. The cost function for partitioning algorithm is $f_o(\pi, E)$ where E represents the hyperedges. In our case, cost is the run-time for the Algorithm. The weight function for partitioning algorithm is $f_w(\pi)$ which is the cumulative vulnerability score for each partition in our design.

The weight of each partition can be fetched from a weight function by relation $W_i = f_w(B_i)$. The average vulnerability score of all partitions is W_{avg}. The goal of the algorithm is to optimize f_w and f_o, such that $W_i < (1 + \epsilon)W_{avg}$ [30].

In Algorithm 2 we check all vulnerability and reachability

Algorithm 2 Sub AG Generation Algorithm

```
1: procedure GENERATE-SUBAG(SHG)
2:     EN = Find_Edge_Nodes(SHG);
3:     for each e ∈ EN do
4:         Attackers = Find_External_Nodes_with_priv(e)
5:     Vulns = Find_Vulns_Info(SHG)
6:     Reaches = Find_Reach_Info(SHG)
7:     SAG = Create_SubAG(Attackers, Vulns, Reaches)
8:     while true do
9:         PostConds = Find_New_PostConds(SAG)
10:        for each p ∈ PostConds do
11:            Write_To_RDD(Agent, p)
12:        PostConds =Read_NewPostCond(Agent)
13:        if PostConds.size() == 0 then
14:            break
```

edges. The preconditions of the AG are the requirements for an exploit to be successful. The postconditions on the other hand mean privilege gained on successful exploit e.g. root access. Since postconditions act as trigger or precondition for another vulnerability, all new post conditions are written to the RDD. The agents or processors can check post-conditions specific to their SubAG and update the SubAG if necessary. The partitions generated from Algorithm 1 are taken as input SHG for Algorithm 2. Since we check the RDDs for updated information the AG generation process makes sure new information such as nodes leaving the system or VMs migrating from one Physical server to another are reflected in real time.

Next, in Algorithm 3 we merge the individual SubAGs. The input for the Algorithm is an array of individual SubAG's. The edges that are part of bipartite graph G' between two SubAG contain edges such that $e_i \in \{SubAG[v_i], SubAG[v_j]\}$ and $SubAG[v_i] \neq SubAG[v_j]$. This means bipartite Graph $G' = SHG_i \cap SHG_j$. We make use of RDD to update edges that belong to such bipartite matching. Since there will be only a finite number of edges which will be part of such matching, we will be able to do the updates in constant time.

Next we will describe Countermeasure selection algo-

Algorithm 3 Attack Graph Merge Algorithm

```
1: AG ← ∅
2: procedure GENERATE-FULL-AG(SubAG[])
3:     AG ← SubAG[]
4:     for each e_i ∈ Read_From_RDD do
5:         if e_i = {SubAG_i, SubAG_j} i ≠ j then
6:             AG ← AG ∪ e_i
```

rithm which will select best countermeasure possible from pool of countermeasures based on the current state of network.

4.2 Countermeasure Selection and Deployment

In this algorithm, we consider two tables, one for virtual machines and another for physical servers. Based on a designed vulnerability score, we trigger the migration of one of the VMs. The score itself is the exponential average of the CVSS Base Score from each vulnerability in the VM with a maximum value of 10.

Let VS_i vulnerability score for VM_i is defined as [7]:

$$VS_i = Min\{10, \ln \sum \exp^{BaseScore(v)}\} \quad (1)$$

We use the greedy Algorithm 4 based on threshold CVSS [2] score to perform migration pro-actively, assuming PS = (PS_1, PS_2,...,PS_N); and VM = (VM_1, VM_2,..., VM_K).

The algorithm, pro-actively reshuffles the network re-

Algorithm 4 VM Migration Countermeasure Selection Algorithm

```
   N The number of physical servers
   K The number of VMs
   VS The list of vulnerability score for VMs
1: for i:1 to N do
2:     SP ← PS_size Get physical server Size
3:     while SP > 0 do
4:         max ← k
5:         for j: k+1 to K do
6:             if VS[j] > VS[max] then
7:                 max ← j VM_j has a higher vulnerability
   score VS_j
8: VM_j should be migrated to another Physical server
9:                 SV ← VM_size
10:                if SV < SP then VM_j is in Migration
11:                    SP ← SP − SV
```

sources, thus reduces the attack probability of the attacker. While there are reactive solutions for protecting network resources, like Signature based IDS, deep packet inspection, etc., they can be combined with proactive countermeasure specified by algorithm to further reduce attack scenarios.

Since any change in the network configuration introduces new flow rules into the SDN environment, as well as

alters the AG by disconnecting certain attack paths, while opening up another potential attack paths; two sets of actions follow: *a*) ensure that the new flow rules being injected into the environment satisfy system policies and do not conflict with existing rules; and *b*) generate new AG for the modified system/network configuration.

4.3 Policy Conflict Resolution

The flow rule conflict detection and resolution module as shown in Figure 6 helps resolve conflicts in flow rules in the SDN environment. It consists of several sub-modules that together examine a candidate countermeasure (selected by the Countermeasure selection module), and achieve a conflict free flow rule table that then be used to implement the updated security and traffic management countermeasure. These are:

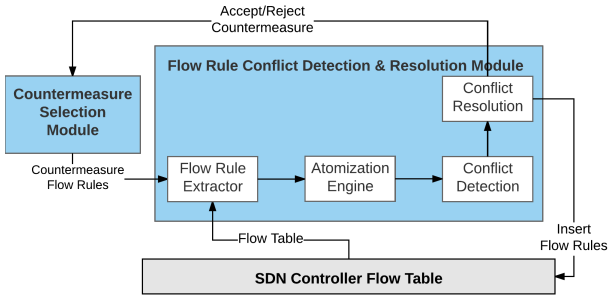

Figure 6: System overview representing flow rule conflict detection module.

- *Flow rule extractor* that receives the candidate countermeasure in flow rule format from the countermeasure selection module, and also obtains the current flow table from the SDN controller.
- *Atomization engine:* Since OpenFlow enables chained flow tables with rules having `set-field` actions, in order to correctly identify conflicts between flow rules, we atomize the flow rules by processing the chains and ensuring that only the atomic actions of permit, deny, and traffic shaping remain. The atomization process itself follows along the lines of `ipchain` processing in Unix.
- *Conflict detection module* that identifies conflicts based on the categories described in Section 3.4.
- *Conflict resolution module* that attempts to resolve conflicts automatically. However, in case of interpretative conflicts (generalization, correlation and imbrication), the candidate countermeasure is rejected and an alternate countermeasure is sought from the countermeasure selection module.

The candidate flow rule and the flow table are on the controller are parsed, and atomized. As a result of this process, every flow rule that had an action which led to the execution of a different flow rule table was modified into one (or more) rules that had terminal actions of permit, deny and QoS. The resulting master flow rule table is then sorted by descending order of the priority before being sent to the conflict detection engine [20]. A Patricia trie [19] based search structure is used to conduct an efficient search to find overlapping layer-3 address spaces.

4.4 SDN Controller Attack Prevention

SDN controller is a critical component in our infrastructure. The failure of the controller can affect the whole system. A distributed controller approach has been discussed in Elasticon [9] where failure of one component will not affect the whole system. Another example of distributed controller has been discussed in Hyperflow [26]. The problem with these approaches can be coordination between various components of the controller. We plan to leverage fault tree [32] based failure analysis technique to address events/preconditions as defined in Section 3.3 that can lead to controller failure. After the fault tree for these events has been constructed we can implement countermeasures to prevent preconditions leading to these events. As software related security bugs can also cause failure events we plan to perform Static Analysis (SA) of controller code for OpenDaylight controller that we are using in our implementation. This can help us in identifying critical software warnings [10] to help controller code developers follow safe coding practices.

5. EVALUATION

To evaluate our system, we considered vulnerability information in MulVAL format, network reachability information, and OpenDaylight based flow rules. The data for MulVAL is generated using vulnerability scans conducted on a prototypical virtualized cloud computing platform based on Openstack framework. The flow rules from SDN controller and candidate countermeasures are translated to MulVAL host access control based rules like `allow(Everyone, read, webPages)`, and `(Src, Dst, Protocol, DstPort)`.

5.1 Complexity Analysis

The complexity of our work is analyzed by individually considering the complexities of AG creation, countermeasure selection and policy conflict resolution and any new postconditions.

Serial partitioning approaches such as KL-Partitioning [16] have a run time of $\mathcal{O}(N^2 \log N)$ to partition hypergraph with N nodes. We employ parallel partitioning algorithm [17] which distributes the task of partitioning hypergraph over p processors, which takes $\mathcal{O}(N/p)+\mathcal{O}(N \log p)$ time to partition a hypergraph with N nodes. In the worst case scenario we have full connectivity between all nodes in the network.

RDDs as discussed in our algorithm contain partition information once the partitioning algorithm finishes its computation. These data structures update any new information from changes due to implementation of a countermeasure in network and use it to update attack graph dynamically. The merge time using RDD in such a scenario will have a new postcondition generated on each node in the network. This allows each node in network to exploit every other node. Using p processors the load of creating all these new post conditions can be distributed over p processors. The AG merge time in such a case is $\mathcal{O}((N/p)^2)$. This is similar to a search using Depth First Search (DFS) over the network, but with the search time distributed over p processors. Thus complexity of AG creation in worst case is sum of hypergraph partitioning time and merge time giving us a run time of $\mathcal{O}((N/p)^2) + \mathcal{O}(N/p) + \mathcal{O}(N \log p)$.

We analyze the complexity of the policy conflict detection

separately below since it involves additional factors such as flow rules and vulnerabilities. If n is the number of flow rules that exist in the environment, r is the number of rules generated by candidate countermeasures, and $|V|$ is the number of vulnerabilities in network and $|CM|$ the number of countermeasures, policy conflict detection and countermeasure selection takes $\mathcal{O}(|V| \times |CM|) + \mathcal{O}(n.r)$.

Thus, the total running time for the algorithm will by combining running times from the hypergraph partitioning, AG merge and the policy conflict detection and countermeasure selection giving a total run time of $\mathcal{O}((N/p)^2) + \mathcal{O}(N/p) + \mathcal{O}(N \log p) + \mathcal{O}(|V| \times |CM|) + \mathcal{O}(n.r) \approx \mathcal{O}((N/p)^2)$.

5.2 Experimental Validation

The experiments were run on an Intel i7 based cloud system. The host Operating System was Ubuntu 14.04. First, we evaluated the dependency of time to generate AG to the number of hosts and their connectivity. We conducted the test on a MulVAL file with $1,700$ densely connected hosts. The implementation of AG clusters/partitions generation involves use of the PySpark based parallelization framework. Each SubAG is generated by one processor in a Apache Spark framework. We computed the time to construct a scalable AG for a fixed number of partitions while increasing number of nodes from $1,700$ to $11,500$. The time to construct scalable AG increases as expected, going from around 9 seconds for $5,600$ nodes to about 28 seconds to generate AG for $11,500$ nodes. This drastic increase in time is because the graph is densely connected and the number of interconnections increase rapidly as we scale up the number of nodes. The time to construct an AG is still less than half a minute, which is acceptable considering the number of reads/write operations involved and merging phase of the different AG partitions.

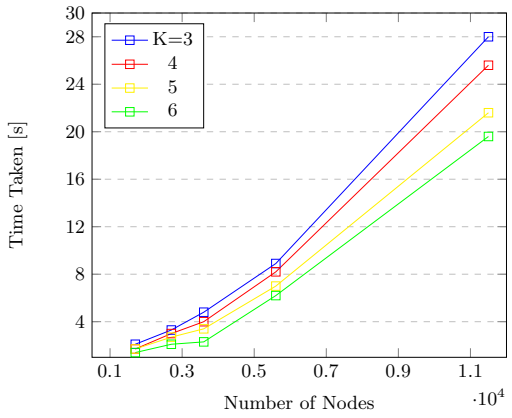

Figure 7: AG generation time vs Number of nodes

Next, we conducted an experiment to measure the time required by the various parts of the algorithm, such as time to partition hyper graph (`PartHypergraph`), time taken for the SubAG generation (`Generate-SubAG`) and time for merging the smaller sub hypergraphs into a large connected AG (`GenerateFullAG`). The results are shown in the Table 5.2. The size of AG is $N = 10,000$. The network is sparsely connected. The time taken by various phases of algorithm doesn't include the time taken to create the input SAG

Parts K	HG Part	SubAG Gen	SHG Merge	Total Time
3	0.30	6.28	0.24	6.82
4	0.35	7.76	0.29	7.914
5	0.36	7.84	0.43	8.27
6	0.37	10.16	0.48	10.64
7	0.38	7.8	0.44	8.25
8	0.36	10.09	0.57	10.56
9	0.36	10.32	0.68	10.8

Table 1: Time based complexity analysis of algorithm

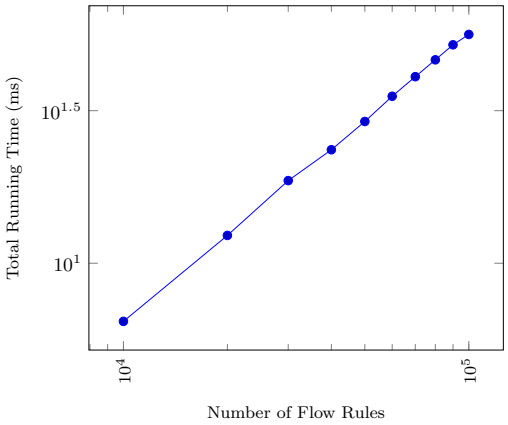

Figure 8: Running time evaluation.

which is produced by MulVAL. We start by generating a Hypergraph, which is dependent on the number of hyperedges. The time taken to generate hypergraph will be fixed if the number of nodes do not change. The SubAG generation takes the most amount of time, and this time increases with the increase in number of partitions K, from $K = 3$ to $K = 6$ which can be attributed to the time involved in spawning separate agents for each SubAG. For $K = 7$ the time decreases for SubAG generation, which is due to the performance gain obtained due to parallelization. For $K = 8$ and $K = 9$, the SubAG generation time increases again. The suitable K can thus be selected based on numerous iterations of the algorithm. The merge time for algorithm shows a constant increase, which is expected except for an outlier at $K = 7$. The overall performance gain in this network configuration due to use of spark based distributed structure is significant. Thus, the solution can serve as the best model for large, densely or sparsely connected networks.

The scalability and the runtime for the flow rule conflict module was tested with an input file containing about $10,000$ atomic flow rules, the processing time was about 6.45 ms. Rules were further replicated and inserted into the system to observe the growth of computation time. Figure 8 shows results from our experiment runs using different input flow table sizes. Ten different test runs each were conducted on flow tables of size varying from $10,000$ to $100,000$ rules, and the resulting running times were averaged to get the results in the plot. The results clearly show that our system effectively identifies flow rule conflicts and takes corrective action in spite of the large data sets. The results also clearly show a $\mathcal{O}(n)$ running time. The run time of approximately 0.56 ms per $1,000$ flow rules for our system bodes well since it clearly indicates that this module would not be the performance bottleneck in an SDN based MTD system.

6. RELATED WORK

A Hierarchical Attack Representation Model (HARM) [11] is used to assess the effectiveness of an MTD technique [12]. While the authors employ several MTD techniques such as OS diversification and address randomization, their experimental results only consider few hundred nodes. The authors do not evaluate if the implemented changes conflict with other security policies. While an implicit assumption of countermeasure not conflicting with current system rules may well be fair in a traditional environment, in a multi-tenant shared data center environment, any change in logical system configuration for a tenant may impact other tenants, and hence evaluating if the implemented countermeasure conflicts with the rules present in the system is essential.

Greedy shuffling MTD strategy using secret proxies to fend of DDoS attack is used in [21] focusing mainly on DDoS attacks. We plan on studying the role of vulnerability in the target environment and cascading dependency created by multi-hop attack in the system.

Predictability oriented defense against adaptive adversaries [22] combines game theory and machine learning. Authors model attackers action in ML feature space to provide defense against current and future attacks. Spam filtering has been used as a target application for this work.
The paper [31] has discussed switching strategies using game theoretic approach for web applications.

Enforcing a conflict free security policy in cloud environments based on SDN has been studied in Flowguard [14], Pyretic [18] and FRESCO [29]. These works deal effectively with direct conflicts by rejecting the policy and implementing role-based and signature-based enforcement to ensure applications do not circumvent existing security policy. However, a flow can be defined in multiple layers, where traditional policy checking approaches do not consider; for examples, indirect security violations, partial violations or cross-layer conflicts cannot be handled them. Moreover, they appear not to fully leverage the SDN paradigm that lets flow rules do traffic shaping in addition to implementing accept/deny security policy. Our previous work [20] overcame some of these shortcomings by including analyzing traffic rate limiting policies along with security policies to detect and resolve direct, indirect and partial conflicts.

7. CONCLUSIONS

We provided a secured cloud framework for MTD solution. As can be seen from the experimental results our solution can scale well on a large network. The DAC methodology ensures that any security threats present in the network are detected and resolved in real time fashion. Our scalable AG solution is very useful in analyzing the security state of a large network, which would otherwise be difficult to interpret for a network administrator. Once we reconfigure the network using a countermeasure selection module, we ensure that there is no security policy violation or conflict in the adjusted network. Our experimental study used OpenDaylight controller and Mininet testbed for evaluation.

The presented work focuses on known network attacks and vulnerabilities to establish attack analysis model and guide system to deploy selected countermeasures to provide MTD solution. Our next step will incorporate anomaly detection models, and the research goal is how to use the programmable networking approach to improve their detection accuracy. For example, changing the network filtering and traffic forwarding rules can help the system to reduce the unrelated network flows for traffic-pattern based anomaly detection models. While signature based detection tools show good performance, identifying known attacks like TCP SYN-Flood, ICMP-flood. Although anomaly based detection models have high false positive rates sometimes, they can help identify attack patterns not yet seen such as in zero-day attacks. We plan to employ regression and other statistical measures to ensure accuracy of anomaly detection methods. We also plan to deploy our solution in a full-fledged Openstack based cloud environment.

Acknowledgments

This research is supported by NSF Secure and Resilient Networking (SRN) Project (1528099) and NATO Science for Peace & Security Multi-Year Project (MD.SFPP 984425). S. Pisharody is supported by a scholarship from the NSF CyberCorps program (NSF-SFS-1129561).

8. REFERENCES

[1] Mininet virtual network, 2015.
[2] CVSS, 2016.
[3] PySpark, https://spark.apache.org/docs/0.9.0/python-programming-guide.html, 2016.
[4] E. S. Al-Shaer and H. H. Hamed. Firewall policy advisor for anomaly discovery and rule editing. In *Integrated Network Management, 2003. IFIP/IEEE Eighth International Symposium on*, pages 17–30. IEEE, 2003.
[5] C. Baier, J.-P. Katoen, and K. G. Larsen. *Principles of model checking*. MIT press, 2008.
[6] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM, 2007.
[7] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang. NICE: Network intrusion detection and countermeasure selection in virtual network systems. *Dependable and Secure Computing, IEEE Transactions on*, 10(4):198–211, 2013.
[8] C.-J. Chung, T. Xing, D. Huang, D. Medhi, and K. Trivedi. SeReNe: On establishing secure and resilient networking services for an sdn-based multi-tenant datacenter environment. In *Dependable Systems and Networks Workshops (DSN-W), 2015 IEEE International Conference on*, pages 4–11. IEEE, 2015.
[9] A. A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella. Elasticon: An elastic distributed SDN controller. In *Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems*, pages 17–28. ACM, 2014.
[10] S. Heckman and L. Williams. On establishing a benchmark for evaluating static analysis alert prioritization and classification techniques. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 41–50. ACM, 2008.

[11] J. B. Hong and D. S. Kim. Performance analysis of scalable attack representation models. In *Security and Privacy Protection in Information Processing Systems*, pages 330–343. Springer, 2013.

[12] J. B. Hong and D. S. Kim. Scalable security models for assessing effectiveness of moving target defenses. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 515–526. IEEE, 2014.

[13] W. House. Trustworthy cyberspace: Strategic plan for the federal cyber security research and development program. *Report of the National Science and Technology Council, Executive Office of the President*, 2011.

[14] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao. FLOWGUARD: building robust firewalls for software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 97–102. ACM, 2014.

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

[16] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Applications in vlsi domain. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 7(1):69–79, 1999.

[17] G. Karypis and K. Schloegel. Parallel graph partitioning and sparse matrix ordering. *University of Minnesota, Department of Computer Science and Engineering*, 2013.

[18] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker, et al. Composing software defined networks. In *NSDI*, pages 1–13, 2013.

[19] D. R. Morrison. Patricia - Practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM (JACM)*, 15(4):514–534, 1968.

[20] S. Pisharody, A. Chowdhary, and D. Huang. Security policy checking in distributed SDN based clouds. In *2016 IEEE Conference on Communications and Network Security (CNS) (IEEE CNS 2016)*, Oct. 2016.

[21] A. S. Quan Jia, Kun Sun. Motag: Moving target defense against internet denial of service attacks. In *Proceedings of 22nd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9. IEEE, 2013.

[22] K. G. Richard Colbaugh. Predictability oriented defense against adaptive adversaries. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 14–17. IEEE, 2012.

[23] R. Saha and A. Agarwal. Sdn approach to large scale global data centers. *Proceedings of Open Networking Summit, Santa Clara, California, USA*, 2012.

[24] R. E. Sawilla and X. Ou. Identifying critical attack assets in dependency attack graphs. In *European Symposium on Research in Computer Security*, pages 18–34. Springer, 2008.

[25] B. Schmerl, J. Cámara, G. A. Moreno, D. Garlan, and A. Mellinger. Architecture-based self-adaptation for moving target defense. Technical report, Technical Report CMU-ISR-14-109. Carnegie Mellon University, 2014.

[26] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao. Are we ready for SDN? implementation challenges for software-defined networks. *IEEE Communications Magazine*, 51(7):36–43, 2013.

[27] O. Sheyner and J. Wing. Tools for generating and analyzing attack graphs. In *Proceedings of Vol. 3188 Lecture Notes in Computer Science pp 344-371*, pages 344–371. Springer, 2003.

[28] O. M. Sheyner. Scenario graphs and attack graphs. *PhD Thesis, CMU*, 2004.

[29] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson. Fresco: Modular composable security services for software-defined networks. 2013.

[30] A. Trifunovic. *Parallel algorithms for hypergraph partitioning*. University of London, 2006.

[31] S. G. Vadlamudi, S. Sengupta, S. Kambhampati, M. Taguinod, Z. Zhao, A. Doupé, and G. Ahn. Moving target defense for web applications using bayesian stackelberg games. *CoRR*, abs/1602.07024, 2016.

[32] B. Vesely. Fault tree analysis (FTA): Concepts and applications. *NASA HQ*, 2002.

[33] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.