FAPA: Flooding Attack Protection Architecture in a Cloud System

Kazi Zunnurhain

Department of Computer Science
PhD Candidate
SEC 3429
The University of Alabama
Box 870290
Tuscaloosa, AL 35487-0290
E-mail: kzunnurhain@crimson.ua.edu

Susan Vrbsky

Associate Professor
SEC 3430
The University of Alabama
Box 870290
Tuscaloosa, AL 35487-0290
E-mail: vrbsky@cs.ua.edu

Ragib Hasan

Assistant Professor
University of Alabama at Birmingham
CIA-JFR Facebook Suit
1300 University Blvd, AL 35294
E-mail: ragib@cis.uab.edu

Abstract: The rate of acceptance of clouds each year is making cloud computing the leading IT computational technology. While cloud computing can be productive and economical, it is still vulnerable to different types of external threats, one of which is a Denial of Service (DoS) attack. Taking the cloud providers' security services could cause disputes and involvement of hidden costs. Rather than depending on cloud providers, we have proposed a model, called FAPA (Flooding Attack Protection Architecture), to detect and filter packets when DoS attacks occur. FAPA can run locally on top of the client's terminal and is independent of the provider's cloud machine. In FAPA, detection of denial of service is accomplished through traffic pattern analysis and it removes flooding by filtering. Both in the cloud and on the cluster, our experimental results demonstrated that FAPA was able to detect and filter packets to successfully remove a DoS attack.

Keywords: DDoS, Cloud, Behavioral Pattern, Filtering, Bandwidth, Throughput, Transfer Packets.

Reference to this paper should be made as follows: Zunnurhain, K., Vrbsky, S.V. and Hasan, R. (xxxx) “FAPA: Flooding Attack Protection Architecture in a Cloud System”, Int. J. Cloud Computing, vol. X, No. Y, pp. xxx-yyy.

Biographical notes: Kazi Zunnurhain received his B.Sc. Degree in Computer Science from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2006. He received his M.Sc. degree in Computer Science from University of Alabama, Tuscaloosa, AL, in 2011. Now he is enrolled as a PhD student in Department of Computer Science at the University of Alabama. His research interest is security issues in cloud computing.

Susan V. Vrbsky is an Associate Professor of Computer Science at The University of Alabama. She received her Ph.D. in Computer Science from The University of Illinois, Urbana-Champaign. She received an M.S. in Computer Science from Southern Illinois University, Carbondale, IL and a B. A. from Northwestern University in Evanston, IL. She is the Advisor of the Cloud and Cluster Computing Lab. Her research interests include database systems, data management in clouds, green computing, data intensive computing and database security.

Ragib Hasan is an Assistant Professor of Computer and Information Sciences at the University of Alabama, Birmingham. He received his Ph.D. in Computer Science in October 2009 from University of Illinois, Urbana-Champaign. He is leading the SECuRE and Trustworthy Computing Lab (SECRETLab). He is also the founder of Shikkhok.com- an award winning free education platform in Bengali language and the coordinator of the Bangla Braille project, aimed at creating educational material for visually impaired children in South Asia. His research is funded by grants from the Department of Homeland Security, the office of Naval Research, a 2012 Google Faculty Research Award, and a 2012 Amazon Research Grant. His research interests include cloud security, secure location provenance, secure data provenance, data waste management.

1 Introduction

Cloud computing is the realization of the concept, *Utility Computing,* in the real world, providing on-demand access to services ranging from hardware to platform to software services levels (Khorshed et al. 2012). Cloud infrastructure is well equipped to satisfy on-demand service requirements. The dynamic features of a cloud boost the demand for cloud systems in IT industries as well as small business companies. Industries are not required to plan for their IT growth in advance with the “pay as you go” feature of a cloud system. Instead, users of a cloud only need to request and pay for services as they are needed. Users of a cloud can range from single users taking advantage of the benefits of a public cloud to multiple users in a startup company utilizing a commercially rented public cloud. For example, a single user can use the cloud to store his/her personal data files or image files. A startup company can utilize the infrastructure provided by a public cloud, such as a Visual Studio framework, to develop a website consisting of multiple types of databases in the environment of the cloud provider. Utilization of a cloud for a range of services has been shown to be very cost-effective (Anderson, *et al.* 2010; Kodada, *et al.* 2012).

The main properties that convert a data center into a cloud are: virtualization, scalability, elasticity, load-balancing, pay-as-you-go and availability (Zunnurhain, *et al.* 2011). With virtualization, users can run their instances with a variety of application options provided by the cloud provider, or run the user’s chosen applications. Virtual machines (VMs) can be of variable size depending on the customer’s application requirement. VM instances run on top of a monitoring system independent of the operating system in the local machine. Scalability is a feature in which the cloud can scale up or down to satisfy an increasing in demand from the user. The feature of elasticity allows for an increase or decrease in capacity, with the cloud provider dynamically adding hardware as needed for a sudden increase in demand. Similarly, the load balancing feature provides the cloud with a certain extended capability, such as replacing an overloaded server or faulty system with another server and reallocating the pending tasks of the faulty server to the newly assigned server (Jenson *et al.* 2009 and Zunnurhain *et al.* 2011). Pay-as-you-go is the basic feature of cloud computing, in which cloud users pay for what they have used. A customer is charged depending on the total time of use or amount of data stored. Availability is another key feature, as users are paying for cloud services.

Although cloud computing is emerging into the IT market with much potential, a great deal of work is still needed in the domain of security to minimize threats (Mansfield, 2011; Bedi, et al. 2012; Khorshed et al. 2012). Many small or midsized organizations adopt cloud computing to reduce the upfront investment costs, to minimize maintenance work in its IT infrastructure, and to enhance on-demand capabilities. However, there is a risk for not doing an assessment on security when utilizing a cloud. If the information stored on a cloud is compromised, then not only can money be lost but personal information can also be stolen.

Cloud systems can be compromised by different kinds of security attacks, including denial of service (DoS) or distributed denial of service (DDoS) attacks. The basic property of a cloud is to provide highly available and uninterrupted services for cloud users, and DoS/DDoS are the kinds of attacks that can cause service interruptions. Hence, it is important for the cloud provider and users to have a mechanism in place to protect users against this type of attack. In this paper we propose one such strategy for protection against DoS and DDoS attacks.

There are different types of DoS and DDoS attacks that can occur in traditional systems. Ping flooding is a basic type of flooding of the targeted system (Web server) and is accomplished by sending ping commands. Some of the packets are discarded by high capacity links but the remaining links clog most of the capacity on the target link. In this traditional type of flooding the source needs to use its IP address to generate the ICMP echo request, which obviously exposes the attacker. Instead, spoofing is employed by the attacker, which is the initial step in the DoS chain of events. Spoofing is a mechanism where the attacker camouflages his identity behind the IP address of a legitimate user or an unused IP address of a domain that has been unused for a long time (Stallings, et al. 2011). The adversary then generates a bulk amount of fake requests and sends them towards the target machine, which is in general the main server of the cloud system. Before reaching the server, validation of these packets is undertaken. Legitimate requests starve while resources are engaged in validating the spoof requests. Eventually DoS takes place, due to the enormous number of spoof packets thrown by non-legitimate users. In a large-scale distributed system, like a grid or a cloud, if the same types of attacks take place from many network terminals targeting a single server in the distributed system, this results in a DDoS. The entire distributed system can be compromised by a single point attack due to spoof packet propagation (flooding).

IP spoofing attacks impose substantial threats for Internet services. According to some surveys it has been found that every year there are about 4000 to 7000 DoS attacks taking place in different network places, such as a public website (yahoo, Facebook), private sectors (Company websites: Aramco), financial institutes (Banks) and many more (Duan et al., 2006; Muhlbauer et al., 2006; Bremler et al., 2005; Jin et al., 2006). According to Prolexic's Q3 2011 report (one of the first and largest companies offering DDoS mitigation) (Gens, F. et al. 2008), 24% of all cyber-attacks were SYN flooding. The report also stated 22% of ICMP and 19% of UDP attacks were accomplished by cyber adversaries (Mansfield 2011). TCP-SYN packets can clog the victim's bandwidth easily if the attacker is adept enough to exceed the maximum bandwidth capacity of the network channel.

The main properties of a cloud can, unfortunately, enhance a DoS attack and these properties serve to distinguish a DoS attack in a cloud from traditional DoS attacks as follows. Due to the load balancing feature in a cloud, if a single adversary sends spoof packets to a cloud server, once the server becomes overloaded it will offload its validating tasks to the nearest server. The newly assigned server also eventually becomes overloaded and offloads its task to another server, thus propagating the flooding attack over the entire network. The cloud properties of elasticity and scalability provide further disadvantages during an attack. A server overloaded with enormous validation tasks will scale up to engage more of its resources and computation nodes to validate the spoof packets, continuously exhausting each server with its assigned resources. A dynamic adversary with knowledge of the targeted cloud topology and server connections can even try to compromise the complete system by using a botnet and deploy several handlers to compromise a cloud network.

A cloud's use of virtualization can also be exploited by an adversary. There exist many generic tools (nmap, hping, wget etc.) with which a cloud user can estimate the placement of VMs in a cloud with a high probability (Bedi, H.S. *et al.* 2012). Unfortunately, these tools can be used by the adversaries to impair the cloud system by launching various attacks. A generic network testing tool can successfully identify a target virtual machine on the cloud with higher likelihood and instantiate VMs co-resident to the target VM to conduct variety of attacks (Ristenpart *et al.* 2009).

There are many DDoS attack tools like Agobot, Mstream, Trinoo (Shah, *et al.* 2013 and Yu, S. *et al.* 2013) that can also be used against a cloud. For example, Agobot can be used as a backdoor Trojan and/or network worms by establishing an IRC channel to the remote servers. Then a client running user friendly social networking websites can be easily controlled by a single attacker to conduct highly effective and efficient DDoS attacks (Shah *et al.* 2013; Ristenpart *et al.* 2009 and Bedi, *et al.* 2012). Obviously, in these examples of DoS attacks in the pay-as-you-go model of a cloud, users can be charged for services not requested. These observations of DoS vulnerabilities in a cloud environment motivated our work to design a strategy for DoS protection in a cloud system.

While currently there exist many different threats of DoS attacks, there are also different types of countermeasures that promise to protect flooding. As will be discussed in Section 2, existing countermeasures are not cost effective in a cloud and may require an upgrade in the end systems. They cannot protect against flooding, neither proactively nor reactively in the cloud system. In this research, our focus is to design a solution for a cloud system that protects against a DoS attack but has a moderate overhead cost. To this end we propose a model, called FAPA (Flooding Attack Protection Architecture), which detects flooding and filters packets when DoS attacks occur. FAPA can run locally on top of the client's terminal and is independent of the provider's cloud machine.

In order to design the filtering mechanism for FAPA, in this paper we describe the traffic analysis we performed to explore traffic behavior when a cloud is compromised. FAPA deploys filtering based on the traffic pattern learned from the network systems. We begin by analyzing the number and length of incoming TCP-SYN requests and the SYN-ACK responses from the server with and without flooding. Next, we focus on the virtualization feature of clouds by considering the impact of flooding on the bandwidth and transfer rates of sibling and neighbors of a compromised VMs. Information from this traffic analysis is subsequently utilized in the FAPA filtering mechanism to detect spoof IP addresses. Experiments on a private cloud utilizing the filtering mechanism is applied to sample data collected from CAIDA (Cooperative Association for Internet Data Analysis) to measure a large amount of DoS compromised network packets.

The objective of the research in this paper is to provide a model to effectively protect against DoS attacks in clouds. The experiments we perform determine the effectiveness of the FAPA filtering model in detecting and filtering spoof packets. Also, unique to this research, is the study of the effect of flooding from an external adversary on sibling VMs and neighbor nodes, a topic which to the best of our knowledge has not been previously studied.

This paper is organized as follows. Ongoing related research in clouds for DoS/DDoS protection is presented in Section 2. A demonstration of our proposed solution, which is a reactive approach and depends on the running virtual machines, appears in Section 3. This is followed by the illustration of our experimental results and analysis, and the modification of our filtering mechanism in Section 4. We present future directions and conclusions in Section 5.

2 Related Work

In this section we discuss previous research work related to DoS attacks in a cloud environment and their remedies. We consider work that is relevant to different aspects of our FAPA model, such as strategies for detecting intrusions, traffic analysis during attacks, and filtering. We identify the weaknesses of each approach and discuss the feasibility of the solution for DoS or DDoS protection in a cloud environment.

In order to address security issues in the cloud infrastructure, several intrusion detection (IDS) systems have been proposed. Roschke *et al.*in 2009 proposed an architecture to detect DDoS attacks in a cloud system. They identified IDS management issues with respect to both host IDS and network IDS, but do not provide information about how the DDoS attacks were generated and detected in the cloud infrastructure. Another architecture was proposed for detecting intrusion in cloud computing. To detect DDoS attacks a behavioral analysis was performed, including knowledge-based analysis. The authors did not consider the virtualization issue in each physical node, which is one of the most essential characteristics of a cloud system.

Rahmani *et al.* (2008) proposed a scheme for DDoS flooding attack detection in the network layer, based on F-divergence. This scheme identifies a DDoS based on the number of packets and number of connections in the network. For a cloud infrastructure it is not a good solution because a change in the number of IP packets may be due to a spike in the demand by the user or an upgrade in resources by the cloud provider.

Khorshed *et al.*in 2012 have tried to classify DoS attacks in cloud computing using different rule-based learning. They claim to identify attacks and propose a remedy from the client's perspective. The principal idea behind their experiments is to check the CPU performance and network usages of the physical machine. The strategy involves running a statistical analysis among a few classification Machine Learning algorithms already implemented in *Weka*, to observe which algorithm performs the best in detecting the DoS attacks. However, a cloud user is usually unaware of the physical machine on which his/her application is running and it is not unclear how the proposed remedy removed the ongoing attacks after the detection.

In 2010, Huan Liu *et al*. proposed a DoS prevention mechanism for clouds considering a possible new form of DoS attack. Usually, attackers target the uplink capacity as it is smaller than the aggregated capacity, and send spoof packets to other hosts in a remote router. Attackers can learn the network topology by trace routing because of several reasons:

1. for an application running in remote routers, requested packets will occupy less bandwidth than the host router, thus the attacker can make an assumption about the hop-count,

2. an attacker can send a continuous sequence of packets from a number of sources toward a presumed sink, then by analyzing the traffic rate an adversary can make an assumption about the number of switches between the source and the sink (Jenson *et al.*, 2009).

Several countermeasures can be adapted, such as providing full bisectional bandwidth, but it will take years to deploy this kind of a network and replace all the legacy routers from the current network (Liu et al., 2010). Another approach could be capping each server with limited bandwidth, but for a dynamic system like a cloud, it will affect the throughput. Additional charges could be considered for bandwidth consumption, but again it is not economical.

The author’s suggestion in (Liu et al., 2010) was to deploy a monitoring agent. The host will send UDP help packets when an application detects a limited amount of bandwidth for a long time. In such a situation, an assumption is that some of the UDP packets will pass through the router and ask for help from the monitoring agent. The agent will search for an active standby server in other routers to reassign the application in another sub network. The disadvantages to this approach are potential delays, a large amount of message passing, unnecessary help packet generation and additional bandwidth.

In their Source Based Filtering (SBF) mechanism, in 2009 Yi, F. *et al*. were motivated by the reliable reappearance of legitimate IP addresses in a webserver in the traffic analysis for a small stub network, and the stable nature of hop counts of each client from the server. They assume that a spoofed IP address residing in the same address space of the handlers is very unlikely. They maintain an IP address space in the server memory for likely IP addresses and filter any incoming packet whose source address is absent in the address table. In a cloud system, maintaining an address space is not a feasible solution. Also, a co-resident adversary is not addressed in their work.

To protect against DDoS in grid and cluster environments, similar to SBF in 2009, a mechanism was proposed in 2012 by Prasad and Kodada *et al.* In the modification of their mechanism, rather than maintaining an IP address table, both the IP address and Mac address were stored at the beginning of cluster node booting. Their algorithm compares each packet's IP address and Mac address three times with the stored addresses in the cluster head. This strategy requires additional memory space as well as increased time for comparison checking.

In 2012 Bedi *et al.* proposed a game theoretic defense mechanism for securing the cloud infrastructure against co-resident DoS attacks, in which an inside attacker compromises a network channel shared by multiple VMs, co-residing in the same physical node of the cloud system. The proposed mechanism does not provide protection against external adversaries, which is a focus of our research.

While there have been some strategies to address DoS in clouds as described above, none of them provides a comprehensive solution for addressing DoS in clouds. We proposed a solution for heterogeneous cloud systems that does not increase the cost nor decrease the Quality of Service for the users. It will not result in an increase in the network packets as only the most relevant network packets will be compared. Instead of checking the performance of the physical machine on which an application is running, we propose performance and usage measurement checking on the virtual networks, which is interfacing the running VM with the physical network device installed in the client's terminal. Through analysis, FAPA will detect the malicious behavior. In the next section we present the FAPA model.

3 Flooding attack protection architecture

3.1 MODEL DETAILS

To overcome security threats we propose our model, called FAPA, containing different components that collaborate to prevent a cloud system from DoS attacks. FAPA was designed by considering machine learning flows and considering the lower level networking needs required to analyze individual packets. It captures raw packets and through analysis detects the behavioral pattern of traffic packets with respect to individual users in each session. These patterns invoke the server to propagate an announcement to the user and filter the packets if an intrusion is detected.

In FAPA, separate modules have individual functionalities. As shown in Figure 1, the modules in FAPA are as follows:

1. Traffic Unpacking
2. Feature Selection
3. Comparison Checking
4. Validity Checking
5. Profile Generator.

FAPA begins by fetching each incoming network packet from a domain in a packet-by-packet manner. A packet can be TCP, UDP, or IPv4. The packet is unwrapped in the Traffic Unpacking module. In the Feature Selection module, pertinent header information is recorded and used for profile generation. These profiles identify the uniqueness of every running instance in the cloud. If spoofing takes place, then these unique features change.

FAPA also utilizes a second capturing style, in which the throughput of certain packet types is recorded. (A handler is defined to select packets either one by one or in a loop fashion for a bulk amount.) The Comparison Checking module monitors the throughput pattern of each packet type and compares it with the previously recorded packet parameters. Any change in traffic behavior or altering of certain bits in the packets will generate an alarm in the Alarm Generation module. The filtering process consists of calculating the spoof traffic, miscellaneous traffic and original traffic. If a legitimate user is compromised then the attacker will be traced back and packets coming from the actuator will be filtered. Regular services in the cloud will not be affected.

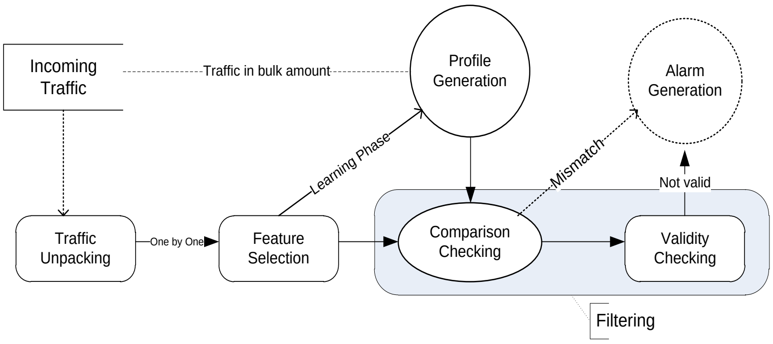

Figure 1: FAPA Model

3.2 FAPA PROTOTYPE IMPLEMENTATION

In this section we discuss how each module works and its significance in protection from flooding.

3.2.1 Traffic Unpacking

A Cloud network can consist of several network devices. In the Traffic Unpacking module the device list is pre-calculated before packet capturing begins. As mentioned earlier, users have privileges to monitor the current traffic situation locally, so users are enabled with device selection to check on the traffic flow.

Once the network device is chosen, different types of packets are captured and dumped in a buffer. TCP-SYN, IPv4, UDP are the major types of packets captured by the module. TCP and IP packets are the most useful types containing the source port, destination port, address of the originator and specific kinds of flag bits, such as SYN, ACK, FIN, ECE, etc. These flag bits, as well as the size of the header and payloads of each packet, contain important information. The intention is to generate the behavioral pattern during the occurrence of denial of service or flooding attacks. As an example, if the FIN bit of a TCP packet is set, then the sender will send no more data because that was the last packet in that session. If the SYN bit is set, it is the first request packet for a TCP connection.

3.2.2 Feature Selection

This module is responsible for fetching packet specific features for behavior analysis. Pertinent header information, such as, header length, window scale, payload size, source and destination ports, source and destination IP addresses, and some flag bits are unwrapped, recorded and used for profile generation after packets are captured. Here the retrieved packet's information is analyzed based on the session time of each user. An application running in the cloud with extensive power of behavior recognition will require no additional devices to guard the system from DoS threats.

3.2.3 Comparison checking

The Comparison Checking module will verify the packet types, their sizes and all infrastructure details with previously recorded information. This module preserves the access rights, encryption, and date/time information for all the users based on every session in a working day. Checking is done before allowing the network packets to be sent to the server. The request is then forwarded to the Validity Checking module.

3.2.4 Profile Generator

After comparison checking, all the individual traffic patterns are stored in the Profile Generator and are sorted by different packet types. In FAPA, all the nodes running a task have a certain amount of buffer size, payload, header length and window scale. These measurements are kept in the profile generator for a certain period of time before they are refreshed for new types. Within the specified packet interval any packet or multiple packets arriving from the Feature Selection module will be compared in the Comparison Checking module with the help of the Profile Generator.

3.2.5 Validity checking

For validation we have not used any complex authentication protocol or encryption method. Neither do we use any kind of CA agent for generating the certificates for each session. Rather we use the behaviors detected in an earlier module. If any change in pattern is detected, then packets arriving from those nodes are filtered. This filtering is committed based on the type and amount of different packet types.

The FAPA model provides a framework for DoS detection that is applicable on different system components, from individual virtual machines, to the hypervisor. In this paper we focus on the FAPA model for individual virtual machines. We have implemented a prototype of the Traffic Unpacking, Feature Selection, Comparison Checking, Profile Generator and Validity Checking modules. We have performed experiments to obtain data to discover the behavioral patterns in virtual machines with and without flooding. The results from our experiments are described in the next section.

4 Experimental Results

In our study, we performed experiments on two different types of platforms, a cluster and a cloud. Initially, we wanted to explore the impact of flooding on typical traffic patterns. We used a cluster to study the traffic behavior under DoS attacks, because the features of a cloud, such as hosting virtual instances, were not needed to obtain results from this experiment. However, in order to compare the impact of DoS on sibling and neighbor VM instances, we ran the experiments on a private cloud. Results from both these types of experiments are incorporated into the FAPA filtering strategy and we subsequently tested the effectiveness of the filtering strategy on our private cloud.

4.1 EXPERIMENTS IN A CLUSTER - METHODOLOGY

The cluster experiments to identify the traffic patterns during flooding were conducted on an Ubuntu 12.04 platform with the enterprise server edition. We utilized the existing cluster in our research lab, in which the server is connected to 10 nodes with an Ubuntu11.10 client. The server was connected through CISCO Linksys E900. All the nodes were connected to the server via Linksys Ethernet gigabit switches, LKS-SG5P, in a star configuration. Each node is composed of a 1.2GHz Celeron CPU, 1 Gb 533 MHz RAM and 80Gb SATA 3 hard drive. MySQL Server (version 5.1.54) was installed on each node.

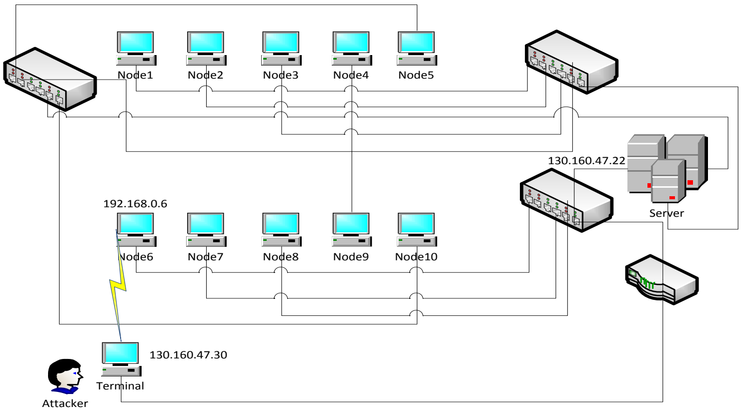

Figure 2: Attempt for spoofing a node inside the cluster

In order to simulate an environment that is flooded, we assumed that an adversary had control over an IP address inside the cluster. Flooding was then generated through a terminal, which accessed the network via the E900 router. The terminal where flooding originated (IP address: 130.160.47.30), used Node6 (IP address: 192.168.0.6) for sending spoof packets to the server as shown in Figure 2.

TCP-SYN flooding was conducted from all nodes in the cluster by targeting making TCP requests for the Ubuntu server to process while the server was busy executing SQL queries. In our experiments, the measurements are kept in the Profile Generator for 1000 packets before they are refreshed for new types. In order to observe the traffic behavior under flooding, we measured the number of TCP reverse and forward packets, and TCP Window scale.

4.2 EXPERIMENTS IN A CLUSTER - RESULTS

4.2.1 Comparison of Packets in an Uncontrolled Environment

In this first experiment we compared the total number of forward and reverse packets in both environments, with and without flooding. Packets were captured with respect to in-flow and out-flow in a bulk amount. The incoming TCP-SYN requests are the forward packets and the SYN-ACK responses from the server are the reverse packets. Figure 3 illustrates the number of forward and reverse packets for the TCP protocol in a non-flooded environment. Figure 4 identifies the number of each packet in a flooded environment. The total number of packets captured increases from 150 to 5000 in Figures 3 and 4.

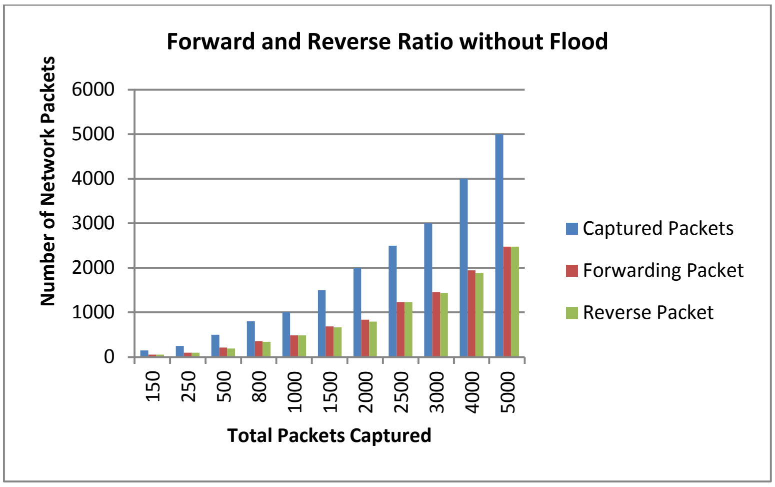

Figure 3: Forward and Reverse Packets without Flooding

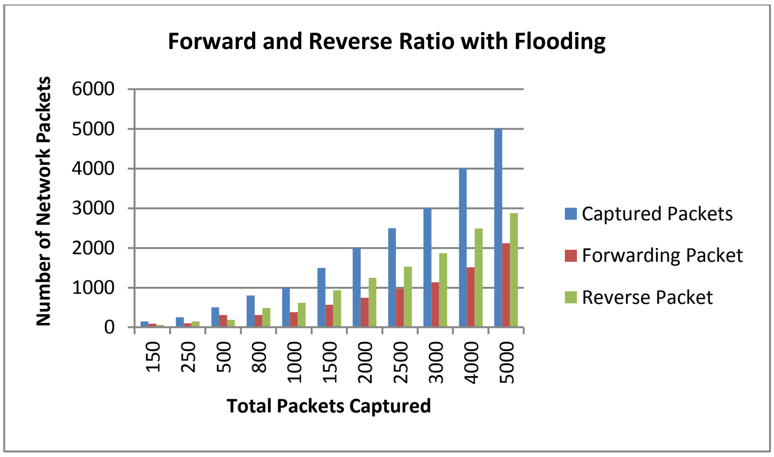

Figure 4: Forward and Reverse Packets with Flooding

In Figure 3 the number of forward packets is almost equal to the reverse packets. This means a user makes a TCP-SYN request, receives the SYN-ACK response from the servers and responds with the ACK message. Our observation is that this is the ideal case in a non-flooded environment. In Figure 4 the number of reverse packets ranges from 15% to 25% when there is flooding, whereas no increment is observed without flooding. According to TCP topology, the amount of SYN-ACK increases only when the server does not receive the final ACK message from the requester. Our observation is that flooding causes an increase in the number of reverse packets, and monitoring if there is an increased number of reverse packets could be one way to identify the impact of flooding on network behavior.

This experiment has demonstrated that we can detect intrusion activity trying to flood the cloud system by comparing the number of reverse packets compared to forward packets. Also, we detected missing packets in a flooded environment, as only TCP packets were allowed to pass from the server.

4.2.2 TCP Window Scale in Variant Environments

In this experiment we observed the effect of flooding on the TCP packet's window length. We captured packets using the loop handler described in Section 3.1, which is designed for continuous capturing and conducted flooding to see if the window scale changed. We performed flooding three times, and we took 5 samples for each scenario and averaged them. Figure 5 illustrates the results of the experiment in which the number of packets is displayed in the horizontal axis and the Window Length is displayed vertically.

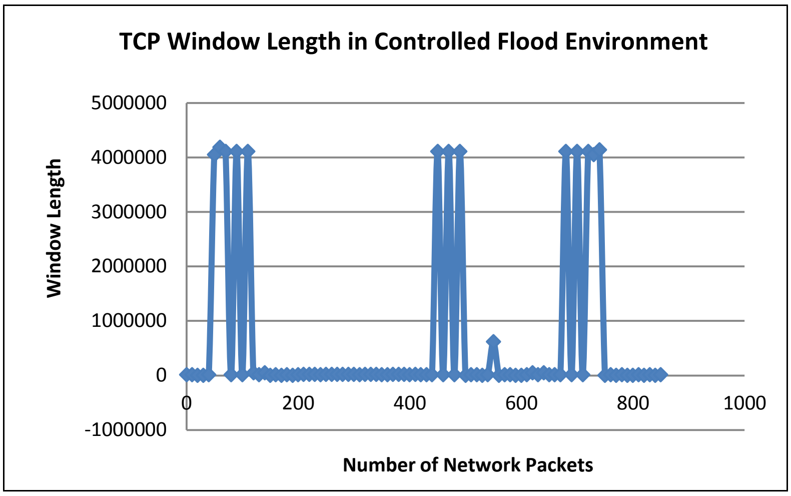

Figure 5: TCP Packet Window Size

As shown in Figure 5, flooding was conducted between 50 and 150 packets, then between 500 and 600 packets, and finally between 700 and 800 captured packets. Our observation in this experiment was beyond our expectation due to the rapid rise in the window length of TCP packets. Also, the periodicity of length change corresponded completely with the flooding events. This traffic behavior can be used to monitor the traffic change in the event of a denial of service.

This experiment has demonstrated that we can detect intrusion activity trying to flood the cloud system by monitoring the window scale change in TCP packets, when the flooding is with TCP-SYN packets. In the next section will describe the details of these experiments in a cloud and the impact of flooding (DoS) on the virtual machine instances in the cloud.

4.3 EXPERIMENTS IN A PRIVATE CLOUD – METHODOLOGY

In order to test the efficiency of our FAPA model to identify and filter, we conducted experiments using an existing Eucalyptus cloud in our research lab. Eucalyptus is a popular open source cloud platform that is compatible with Amazon's EC2 APIs and tools. Eucalyptus also has the advantage of a modifiable hypervisor called, KVM. Important to our experiments, KVM can be used in *vmbuilder* and an SSH connection can be established during the booting of every image. For the cloud we used euca2ools, and there were four OptiPlex machines hosting the Ubuntu servers 12.04 LTS to build the cloud. One machine hosted the cluster controller (CC) with an Intel(R) Core(TM) 2 Duo processor 2.6 GHz speed. Another OptiPlex 755 with an Intel Core 2 Duo with a 2.33 GHz hosted the cloud controller (CLC), the walrus controller (WC) and the storage controller (SC). Both the CC and CLC were connected via a *DLink* Gigabit switch, DGS-2208. An OptiPlex 755 with a 2.83 GHz Intel Core 2 Duo processor hosted the first node controller (NC1). The second node controller (NC2) was hosted on a *Foxconn* motherboard with a 2.6GHz processor speed Intel Pentium(R). Both NC1 and NC2 were connected via the *DLink* Gigabit switch. These two switches were connected to the public network via a Linksys *E900* Cisco router.

NC1 hosted 3 VMs and NC2 hosted 4 VMs. We used KVM and vmbuilder to create each of the VMs with 4GB capacity, a 2GB swap partition and a 4GB variable partition space. For our performance analysis in the cloud, we calculated the bandwidth, transfer rates and throughput of seven VMs. NC1 was interfaced with 3 virtual networks (vnet0, vnet1, vnet2) and NC2 was interfaced with 4 virtual networks (vnet0, vnet1, vnet2 and vnet3).

To conduct the experiment we compromised one virtual machine in each physical node to observe the effect. In Figures 6 and Figure 7, the *blue* bars illustrate the results for each VM before flooding and the *red* bars illustrate the after effect of flooding for each VM.

4.4 EXPERIMENTS IN A PRIVATE CLOUD - RESULTS

4.4.1 Compromising VM6 in NC2

At first we compromised VM6 in NC2 via the external windows terminal. The TCP flooding was conducted from the external terminal connected via E900 with destination address 192.168.1.174. Figure 6 illustrates the results from this experiment.

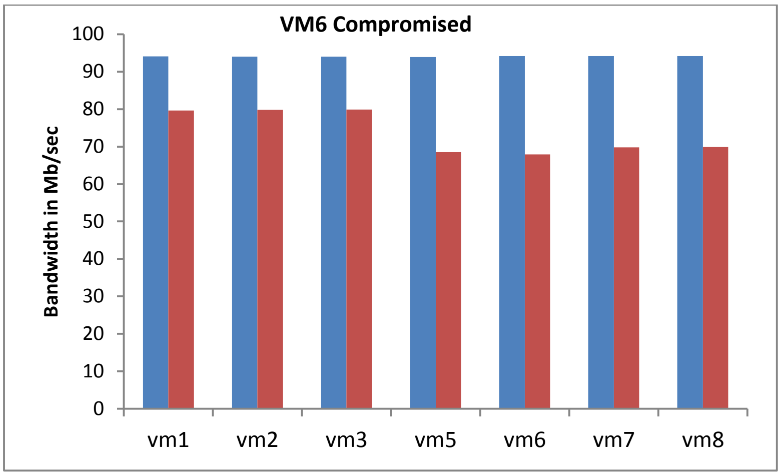

Figure 6: VM6 of NC2 was compromised

In Figure 6, the bandwidth of VM6 along with its sibling VMs was affected the most, as it was the compromised VM. The bandwidth of VM6 decreased from 94.2 Mb/sec without flooding to 67.9 Mb/sec with flooding a decrease of about 29%. The average bandwidth reduction amongst all VMs was 17.34% with flooding. Bandwidth reduction among the siblings (VMs hosted in same node NC2) was about 27% and among the neighbor VMs (VMs hosted in the neighbor node NC1) bandwidth was reduced about 15.17%. NC2 is more affected than NC1.

4.4.2 Compromising VM1 in NC1

Next, to determine if these results would be consistent across each NC, we perform the same experiment by compromising VM1 in the other node controller NC1. TCP flooding was conducted from the external terminal with destination address 192.168.1.250. Figure 7 highlights the impact of flooding.

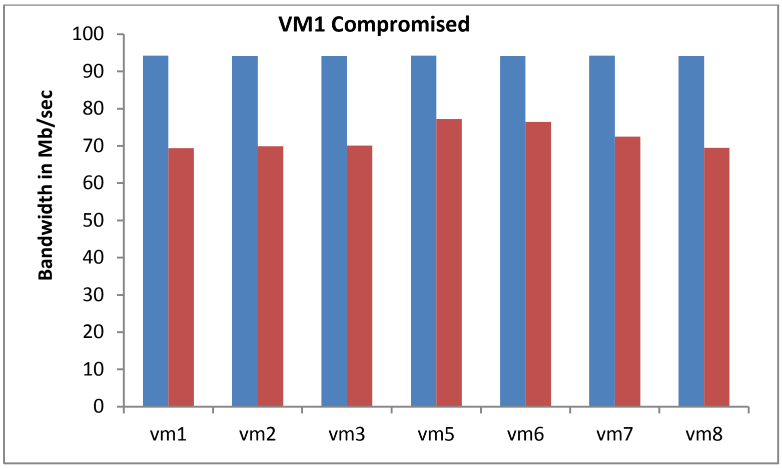

Figure 7: VM1 of NC1 was compromised

In Figure 7, the bandwidth of VM1 decreased from 94.1 Mb/sec without flooding to 69.4 Mb/sec with flooding, about a 26.3% decrease. The average reduction in bandwidth amongst all VMs is 23.53% with flooding. In NC1 the average reduction is 23.9% and in NC2 the reduction average is 21.51%. Again, the NC containing the compromised VM is more affected than the NC without the compromised VM.

The attack on a VM resulted in a reduction in the VM bandwidth that ranged from 26% to 29%. The average bandwidth reduction across all VMs on the same NC as the compromised VM ranged from 24% to 27%. The average bandwidth reduction across all VMs on a different NC as the compromised VM ranged from 15% to 22%.

From the above data we make the observation: *Compromised VMs in a cloud not only affect the co-resident VMs, but the bandwidths of the VMs residing on neighbor nodes are also impacted.*

Although the experimental results are not shown here: *Both sibling and neighbor node transfer rates decreased. However, we were unable to determine if sibling VMs on the same node as the compromised VM are more affected than their neighbor VMs on different nodes.*

4.4.3 Throughput Measurement with bwm-ng

In this next experiment we wanted to see the effect of flooding on throughput. To do so, we used the Linux bandwidth monitor bwm-ng in the node controllers (NC1 and NC2) and flooding was conducted on each VM individually (both NC1 and NC2). Results are shown in Figure 8.

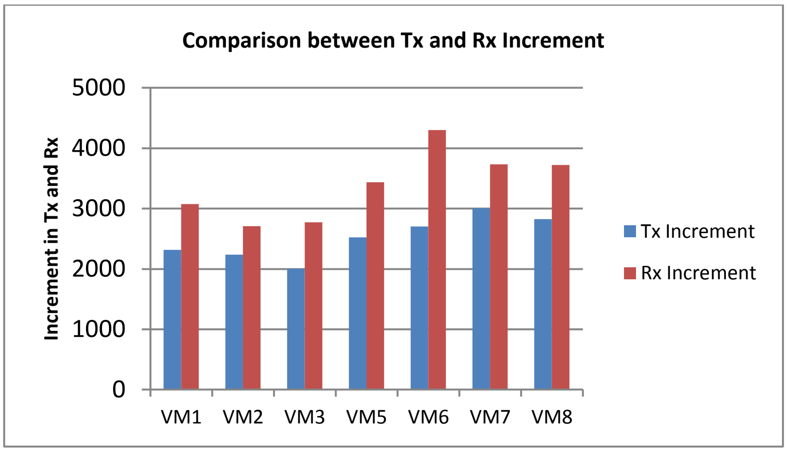

Figure 8: Tx and Rx increment in each VM of the cloud

In Figure 8, the transmission rates increased up to 4299.48 KB/sec and receive rates increased up to 3007.93 KB/sec with flooding. Flooding a VM resulted in an increase 3 times the transmission rate and 10 times in receives rates compared to a non-flooded situation in the cloud for each VM in the cloud.

During our experiments we observed no change in Tx and Rx rates in the non-compromised virtual networks. Therefore, we make the observation: *Flooding from an external attacker on a targeted VM affects the Tx and Rx rates to a great extent on the virtual network of the compromised VM but it doesn't hamper the other virtual networks of the network interface; in other words, co-resident and neighbor vnets are not affected*.

4.4.4 Modified FAPA Filtering

Our initial FAPA filtering scheme was based on threshold values collected from the traffic behavior characteristics during an attack, as described in Section 4.2. We monitor the number of TCP reverse and forward packets and TCP Window scale. After observing the impact of flooding in the virtual networks, we have modified our FAPA filtering mechanism to handle spoof packets to minimize the effect of flooding on virtual networks' parameters.

The FAPA filter algorithm has been modified as follows. Traffic attributes can have variable default thresholds, depending on the size and type of the application running in the cloud user's VM. The FAPA filtering algorithm now generates a profile for arriving packets. Every arriving packet has some nominal values embedded inside the packet which can be utilized to trace spoof packets and filtered when found. The *hop count* parameter for each packet routed through the network is also used to identify spoof packets.

Every packet has a TTL value that identifies the number of hops of each packet. Upon receipt of a packet, the source IP address is extracted from the IPv6 packets. IPv6 has a total of 128 bits. We divide the IP address into n segments, Y_0, Y_1, Y_2 ….Y_{n-1}(see Table I). Each of these segments can have values from 0 to N-1 where N = $2^{32/n}$. The Maximum Hop count can be up to 128 based on the type of OS running in the VM. For the cloud server, all the running nodes provide each segment's information for each hop value.

0	$A_{0,0}$	$A_{0,1}$	$A_{0,2}$	$A_{0,3}$	…	$A_{0,N-1}$
1	$A_{1,0}$	$A_{1,1}$	$A_{1,2}$	$A_{1,3}$		$A_{1,N-1}$
2	$A_{2,0}$	$A_{2,1}$	$A_{2,2}$	$A_{2,3}$		$A_{2,N-1}$
3	$A_{3,0}$	$A_{3,1}$	$A_{3,2}$	$A_{3,3}$		$A_{3,N-1}$
….	….	….	….	….		….
127	$A_{n-1,0}$	$A_{n-1,1}$	$A_{n-1,2}$	$A_{n-1,3}$		$A_{n-1,N-1}$

Table I: IP address segmentation for Hop count permutation

In every row one segment of a hop count is available. An accumulation of 128 segments represents the actual hop count of each IP address. Each entry in the table is represented by $A_{i,j}$ where the value of i ranges from 0 to N-1 (the maximum is 31) and j ranges from 0 to n-1 (the maximum allowable value is 127). All the table entries hold two types of values: one is the current statistics *Present*[i,j] and the other is *Normal*[i,j]. The cloud user's terminal, if compromised by DDoS, adjusts the value of *Normal*[i,j] based on the *Present*[i,j], which is arranged by the incoming packet's source IP address coming from external terminals.

This automatic upgrade of hop values for each packet is affected when there is an outside attacker. Segmentation of a source IP address starts after an attack is detected in the system. A spoof address will not have the coordinated values of each segment because the *Present*[i,j] and *Normal*[i,j] values are altered due to an attack. Once a DDoS attack is detected by the Comparison module of FAPA and an alarm is generated, then all the incoming packets in the victim will be segmented into 32 pieces according to the hop, so the values of each segment can vary. If *f*(Yi) is the function of current and normal statistics, then the score of each segment would vary as follows:

$$f(\mathrm{Y_i})=\begin{cases} Present[i,j]-Normal[i,j] & Present[i,j]>Normal[i,j] \\ \infty & mal[i,j]=0 \\ 0 & Present[i,j]<Normal[i,j] \end{cases}$$

A segment Y_i with any kind of abnormality will be considered as an attack packet. To distinguish the attack packets from normal packets, we used the following norm as a metric by specifying the incoming packet Y as:

$$\|F(\mathrm{Y_i})\| = \sum_{i=1}^{n-1} f(\mathrm{Yi}).$$

The FAPA filtering compares the newly calculated $\|F(Y_i)\|$ with a given threshold μ. $\|F(Y_i)\| \geq \mu * x$, where the smaller the value of x, the stronger the DDoS attack is commencing in the cloud system. The value of x is set globally based on the hop-count packets. Setting an initial value of P_m and Q_m as the current and the ideal profile when the hop count equals to, suppose m, we have $x = P_m/Q_m$. P and Q are globally set parameters for each hop count m based on the node distances from the server, typically depending on the OS type running in the cloud VM (e.g. typically Windows OS has 32 bits to 64 bits, MAC OS has 128 bits, etc.). If x is 0, it will show that no legal packet has arrived before, so we consider the arriving packet with that hop count as an attacking packet. If $x \geq 1$ then the arriving packet is a legal packet and is not in the scope of the hop count attack level. Finally, μ is calculated from the product of the specific segment and the maximum difference between the current and normal statistics as shown below:

$$\mu = n * \underset{i=0, j=0}{\mathrm{MAX}} |Present[i,j] - Normal[i,j]|.$$

As an example, if a packet with a non-legitimate IP address (never appeared before) arrives then *Present*[i,j]>0 but *Normal*[i,j]=0, then $\|F(Y_i)\| = \infty$, and it is considered as an attack packet. But in the case of a legitimate address, that packet will have a valid *Normal*[i,j] value, so $f(Y_i)$ won't be ∞. $F(Y_i)$ is used to calculate the total segment values when a vulnerable packet has arrived in the source, where μ is the threshold to consider and filter an IP address if the limit is exceeded.

Hence, for any kind of external attacks our FAPA filtering will consider the hop count values of the IP address segments and the compromised packets' IP addresses will be conveyed to the rest of the nodes in the cloud through broadcast messages.

4.4.5 FAPA Filtering Experiments – Methodology and Results

In order to test our FAPA strategy, sample data was collected from CAIDA's 2010 DDoS Attack Dataset (Andersen, D., et al. 2010). In this dataset, flooding was generated through ICMP packets. The dataset consisted of about 10 million echo requests and response packets for a server compromised over 24 hours. We calculated the echo requests and the echo responses, and we measured the differences and increments of the response packets per minute. Then we deployed FAPA with the CAIDA dataset, whereby the complete sample file was used in the experiment. The source IP addresses with the destination unreachable responses from the server were considered as malicious users, and those IP packet requests were filtered by the FAPA comparison and validation modules prior to a response from server.

The CAIDA sample data file has a *.pcap* extension, so it was opened in *Wireshark*, a Network protocol analyzer. Based on the *ip.dst*, the captured file was filtered, which actually identifies all the ICMP packets destined to the victim. On average, 350 packets per second were delivered to the victim. Figure 9 illustrates the situation with the help of both ICMP and TCP packets. Only a significant part of the large file is highlighted here. Within 120 seconds about 50k packets were forwarded to the victim of the network. In Figure 9 the red line shows TCP packets and the black line shows ICMP packets.

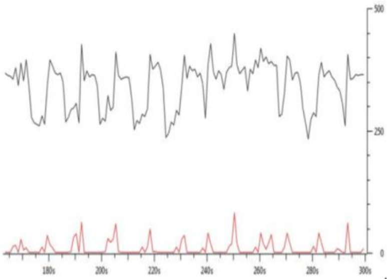

Figure 9: ICMP packets

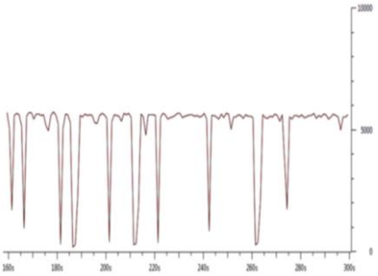

Figure 10: ICMP echo response

The modified filtering, discussed in Section 4.4.4 was applied based on the ip.src (victim's IP address) for analyzing the reply packets from the victim. Figure 10 illustrates the situation after flooding has taken place. It was observed that on average, 4711.8 packets per second were delivered from the victim to anonymous IP addresses for acknowledgement. All these ICMP echo requests were "Destination Unreachable" messages. According to Figure 10 at the time of the capture, about 700k echo responses were propagated from the victim within 120 seconds. Statistically, the response packets were 14 times of the total request packets within the first 120 seconds. After applying the modified FAPA filtering mechanism on the CAIDA captured file we observed significant improvement in filtering the DDoS ICMP packets.

Total Packet Received	9431735
ICMP Destination Unreachable	270105
Bytes Captured	565702045
Bytes Compromised	23581699
Percentage of Compromised bytes	4%
Avg. Mbit/sec	82.8
Avg. Destination. Unreachable Mbit/sec	3.452

Table II: Summary of the Improvement

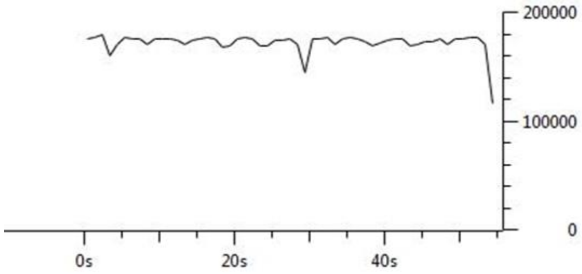

Figure 11: Packets with Filtering

Table II illustrates that the percentage of compromised bytes decreased to 4%. The *Destination Unreachable* packets decreased to 3.4 per second, meaning 408 packets in first 120 seconds. The ratio between forwarded packets towards the victim and response packets from the victim is 1.16. So filtering has successfully blocked incoming spoof packets by 78%, and the response packets from the victim were decreased. Figure 11 highlights the improvements for a portion of the traffic from the captured file. Table II, enhances the visibility of the improvement achieved from the external spoof IP addresses.

Filtering in FAPA causes a drop in compromised packets. By employing the FAPA model on the client's side, users will have complete knowledge of any changes in traffic patterns.

5 Conclusions and Future Work

Results in this paper demonstrated the ability of our proposed FAPA model to detect and filter packets to eliminate a DoS attack. As of now, we have implemented the Traffic Unpacking, Compatibility Checking, Profile Generator and Validation of the FAPA model. All the experiments were conducted based on TCP flooding, and filtering was used to detect and eliminate a DoS attack. Also, we explored the impact of TCP flooding on a compromised virtual machine as well as its siblings and neighbors in a cloud environment. FAPA was able to filter almost 80% of the spoofed TCP packets in a cluster environment, and it also successfully mitigated compromised ICMP echo responses in the CAIDA example (see Table II). FAPA handled the ratio between forward and reverse packets as well as increased the original IP packets by decrementing the spoofed IP packets.

Detection against TCP flooding from only one VM is not enough for modern clouds, because an adversary can flood the system from TCP-SYN packets through a botnet. An adversary can also penetrate the system through UDP packets, HTTP packets, and fake ICMP echo requests. In our future work, we will compromise a virtual machine in a public cloud (such as EC2 or Microsoft Azure) in order to target the private cloud that was used in our earlier experiments. Then we will amplify the impact by deploying an increasing number of instances in the cloud in order to determine if we can successfully accomplish protection against DDoS from many nodes of a public cloud targeting the private cloud. This will also enhance the capability of FAPA filtering in a large scale commercial environment. In the future we will also include experiments with UDP, HTTP flooding. We will check the efficiency of FAPA to eliminate these different types of flooding.

References

Andersen, D., Hick, P. (2010) "The CAIDA (Cooperative Association of Internet Data Analysis) anonymized 2010 Internet Traces". [Online] http://www.caida.org/data/passive/passive2010dataset.xml (accessed January, 2013).

Bedi, H.S., Shiva, S. (2012) "Securing cloud infrastructure against co-resident DoS attacks using game theoretic defense mechanisms". Proceedings of the International Conference on Advances in Computing, Communications andinformatics (ICACCI '12). ACM, New York, NY, USA, 463-469.

Bremler-Barr, A., Levy, H. (2005) "Spoofing Prevention Method".Proceedings of IEEE INFOCOM.

Duan, Z., Yuan, X., Chandrasekhar, J. (2006) "Constructing Inter-Domain Packet Filters to Control IP Spoofing Based on BGP Updates". Proceedings of IEEE INFOCOM.

Gens, F. (2008) "IT Cloud Services User Survey, pt.2: Top Benefits & Challenges". IDC eXchange [online] (http://blogs.idc.com/ie/).

Jin, C., Wang, H., Shin, K.G. (2003) "Hop-Count Filtering:An Effective Defense against Spoofed DDoS Traffic". Proceedings of 10th ACM Conference on Computer and Comm. Security.

Jenson, M., Schwenk, J., Gruschka, N., Iacono, L. (2009) "On Technical Security Issues in Cloud Computing". IEEE International Conference on Cloud Computing.

Khorshed, T., Shawkat, A.B.M., Wasimi, S. (2012) "Classifying different denial-of-service attacks in cloud computing using rule-based learning". Journal-Security and Communication Networks, volume 5, Issue 11 Pages 1235-1247.

Kodada, B., Prasad, G., Pais, A. (2012) "Protection against DDoS and Data Modification Attack in Computational Grid Cluster Environment". International Journal Computer Network and Information Security, Volume 4, Issue 7 July 2012, pp 12-18.

Liu, H. (2010) "A new form of DoS attack in a cloud and its avoidance mechanism". In ACM Cloud Computing Security Workshop (CCSW), pages 65–76.

Muhlbauer, W., Feldmann, A., Maennel, O., Uhlig, S. (2006) "Building an AS-Topology Model that Captures Route Diversity". Proceedings of ACM SIGCOMM.

Mansfield, S. (2011) "DDoS: threats and mitigation". *Journal on Network Security*, Volume 2011, Issue 1, Pages 5-12, ISSN 1353-4858, 10.1016/S1353-4858(11)70128-3.

Rahmani, H., Sahli, N., Kamoun, F. (2012) "DDoS flooding attack detection scheme based on F-divergence". Journal on Computer Communications, Volume 35, Issue 11, 15 June 2012, Pages 1380-1391, ISSN 0140-3664.

Roschke, S., Cheng, F., Meinel, C. (2009) "Intrusion Detection in the Cloud".Eighth IEEE International Conference on Dependable Autonomic and Secure Computing (DASC '09), Chengdu, China, pp.729-734.

Ristenpart, T. and Tromer, E. and Shacham, H. and Savage, S., "Hey, You, Get Off My Cloud: Exploring Information Leakage in Third-Party Compute Clouds," 16th ACM CCS, pp. 199-212, 2009.

Sah, J., and Malik. J., (2013) "Impact of DDOS Attacks on Cloud Environment." *IJRCCT* 2.7 (2013) Volume 2, Issue 7, pp. 362-365.

Stallings, W., Brown, L., (2011) "Computer Security - Principles and Practices", published by Pearson Education, Inc, publishing as Prentice Hall, 2nd Edition, November 19, 2011.

Yi, F., Yu, S., Zhou, W., Hai, J., Bonti, A. (2009) “Source Based Filtering scheme against filtering DDoS attacks”.International Journal on Database Theory and Application, vol. 1, no. 1, pp. 9-20.

Yu, S.; Tian, Y.; Guo, S.; Wu, D., (2013)" Can We Beat DDoS Attacks in Clouds?" *IEEE Transactions on Parallel and Distributed Systems, 2013”*, Volume PP, no.99, pp.1, 1, 0.

Zunnurhain, K., Vrbsky, S. (2011) “Security in Cloud Computing”. Proceedings of the International Conference on Security & Management.

APPENDIX: A

Algorithm 1: SYN Flood DoS with Linux sockets

```
// including the library for memory set
// including standard library for exit (0)
// including errno – for the error number
// including netinet/tcp.h – declarations for TCP header
// including netinet/ip.h – declarations for IP header
Pseudo header:
Construction of a pseudo header for checksum:
for calculation of tcp and ip packets
Construction of a Short CSUM:
        While( number of bytes > 1)
        sum = sum + *ptr++;
        number of bytes = number of bytes – 2;

Flood Method Begin:
Creation of a raw socket
Allocation of a Datagram to represent the packet
Construction of IP Header structure
Construction of TCP header structure:
        Target address is assigned here
Assignment of IP Header fields
Assignment of TCP Header fields:
        Window size is allocated
        Kernel’s IP stack is filled with correct
        checksum during transmission
Construction of IP checksum
Construction of IP_HDRINCL to tell the kernel that
headers are included in the packet
Construction of Loop to flood the target
Loop: (Can be infinite or finite)
        Sending the packets
        if
        Packets containing socket, buffer containing header and data,
        length of the datagram, routing flags and socket   address
        endif
        else
                Print  Error
end Loop

END SYN Flood DoS
```

Algorithm 2: SYN Flood DoS Filtering

```
// including arraylist to keep track of all network devices
// including list to keep the packets as a linked list
// including JPacketHandlerto handle the packet headers
// including PcapPacket – points to the device list
// including protocol – distinguish among TCP, IP, UDP, HTTP headers

Construction of a String Builder:
    Handles the error messages
Construction of ArrayList:
    Listing of all the network devices
    Selection of desire network device for traffic
    Setting flags in Promiscuous mode
    Open one device in Live and start capturing

Defining a JPacketHandler:
    Allocation of memory: tcp, ip, udp, http headers;
     Initialization of counters for different packets;

if packet has header with ip4
    Extraction of source address byte from ip4
    Conversion of byte to string;
    if address matches Original address
        Original_counter++;
        if address matches Suspicious address
            Spoof_Counter++;
            ifSpoof_Counter> Threshold
            go to FilterSetting;
            end if
        end if
    end if

Construction of FilterSetting

// including bpf_program structure.
// instance of a compiled Packet Filter Program.

Initiation of an instance with Bpf Program

    if (compilation of bpf instance, expression, optimize and
    netMask != handler instance.OK)
    print“generate alarm for flooding”
    Propagation of message to all the connected VMs
    Filtering of the originator collected by handler
    end if

END SYN Flood Filtering
```