SDN Oriented DDoS Detection and Mitigation Scheme for Botnet-Based Attacks

Goutham Prasanna
Electrical Engineering Department
San Jose State University, USA
Class ID: 42
goutham.prasanna@sjsu.edu

Nitish Gupta
Electrical Engineering Department
San Jose State University, USA
Class ID: 20
nitish.gupta@sjsu.edu

Sohail Virani
Electrical Engineering Department
San Jose State University, USA
Class ID: 59
sohail.virani@sjsu.edu

***Abstract*—Internet has led to the creation of a digital society, where (almost) everything is connected and is accessible from anywhere. However, despite their widespread adoption, traditional IP networks are complex and very hard to manage. Software-Defined Networking (SDN) is an emerging paradigm that promises to change this state of affairs by separating the network's control plane from the data plane, thereby promoting (logical) centralization of network control and introducing the ability to program the network. However, it has been proven time and again that the SDN is vulnerable to various kinds of attacks like DDoS, DoS, dictionary attacks etc. DDoS attacks mounted by botnets has been termed as biggest threat to internet security today, they target a specific service, mobilizing only a small amount of legitimate looking traffic to compromise the server. Detecting or blocking such clever attacks by only using anomalous traffic statistics has become difficult and devising countermeasures has been mostly left to the victim server. In this paper, an attack detection and mitigation application has been implemented in an SDN environment. Additionally, a mechanism has been developed on the server side to differentiate between legitimate and illegitimate users such that service to former is not affected.**

***Index Terms*— Software Defined Networks (SDN), Denial of Service (DoS), Distributed Denial of Service (DDoS)**

I. INTRODUCTION

The distributed control and transport network protocols running inside the routers and switches are the key technologies that allow information, in the form of digital packets, to travel around the world. Despite their widespread adoption, traditional IP networks are complex and hard to manage. To express the desired high-level network policies, network operators need to configure each individual network device separately using low-level and often vendor-specific commands. In addition to the configuration complexity, network environments have to endure the dynamics of faults and adapt to load changes. Automatic reconfiguration and response mechanisms are virtually non-existent in current IP networks. Enforcing the required policies in such a dynamic environment is therefore highly challenging.

To make it even more challenging, current networks are vertically integrated. Hence, the control plane (that decides how to handle network traffic) and the data plane (that forwards traffic according to the decisions made by the control plane) are bundled inside the networking devices resulting in reducing flexibility, hindering innovation and evolution of the networking infrastructure. The transition from IPv4 to IPv6 started more than a decade ago and still largely incomplete, bears witness to this challenge, while in fact IPv6 represented merely a protocol update. Due to the inertia of current IP networks, a new routing protocol can take 5 to 10 years to be fully designed, evaluated and deployed. Likewise, a clean-slate approach to change the Internet architecture (e.g., replacing IP), is regarded as a daunting task – simply not feasible in practice. Ultimately, this situation has inflated the capital and operational expenses of running an IP network.

A. Software Defined Networks (SDN)

Software-Defined Networking (SDN) is an emerging networking paradigm that gives hope to change the limitations of current network infrastructures [4]. First, it breaks the vertical integration by separating the network's control logic (the control plane) from the underlying routers and switches that forward the traffic (the data plane) [1]. Second, with the separation of the control and data planes, resulting network switches become simple forwarding devices and the control logic is implemented in a logically centralized controller (or network operating system), simplifying policy enforcement and network (re)configuration and evolution. It is important to emphasize that a logically centralized programmatic model does not postulate a physically centralized system [1]. In fact, the need to guarantee adequate levels of performance, scalability, and reliability would preclude such a solution. Instead, production-level SDN network designs resort to physically distributed control planes [4].

Security is expected to be an important application area for the software defined network (SDN). This is because network security requires a close coordination of many network components to defend against an attack [2]. Conventional routers are so difficult to modify their behavior. SDN architecture based on OpenFlow specification makes it much easier to dynamically modify the behavior of network switches [2]. Moreover, in the traditional Internet architecture, routers perform routing and control in a distributed manner. It is hard to elicit an orchestrated network-wide behavior. However, in SDN, the existence of a central controller makes the task so much easier. In this paper, we show that SDN makes it really easy to orchestrate network switches to perform a defensive flow management. In particular, we take the example of distributed denial-of-service (DDoS) attack that relies on a large botnet. By using only the standard OpenFlow interface, a DDoS blocking system can be immediately constructed [2].

B. Distributed Denial of Service (DDoS)

As DDoS attack techniques evolve, effective defense is becoming a challenging task. The representative DDoS attacks today typically target specific services so that only the targeted application is disabled while other network components (e.g. links, switches, routers) are not affected much [1]. Such a targeted attack can easily hide itself in normal traffic due to low required attack intensity. For example, a HTTP GET flooding attack utilizes the vulnerability of HTTP web server implementation (e.g. the maximum number of concurrent connections for HTTP) [1]. In particular, modern DDoS attacks exploit a potentially large number of bots, or compromised hosts. Since these otherwise innocent hosts issue legitimate looking service requests to the attacked server, the attack traffic looks like normal traffic in terms of pps (packets per second), packet size, and packet contents so that it is harder for existing DDoS solutions to block them out from normal packets [1]. In the same vein, signature-based defenses for DDoS attack are not enough to counteract effectively. Also, because the attack traffic volume is small, packet block method based on anomalous traffic statistics does not work, either [1].

C. Botnets

A botnet is a collection of compromised computers often referred to as "zombies" infected with malware that allows an attacker to control them [7]. Computers can be co-opted into a botnet when they execute malicious software. This can be accomplished by luring users into making a drive-by download, exploiting web browser vulnerabilities, or by tricking the user into running a Trojan horse program, which may come from an email attachment. Many computer users are unaware that their computer is infected with bots. Depending on how it is written, a Trojan may then delete itself, or may remain present to update and maintain the modules [7].

D. Honeytokens

"A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource". [5]

Note that in the above definition it is stated that a honeypot does not have to be a computer, but it's a resource that you want the bad guys to interact with. A honeytoken can be a credit card number, Excel spreadsheet, PowerPoint presentation, a database entry, or even a bogus login. Honeytokens come in many shapes and sizes; however they all share the same concept: a digital or information system resource whose value lies in the unauthorized use of that resource. Just as a honeypot computer has no authorized value, no honeytoken has any authorized use. Whatever you create as a honeytoken, no one should be using or accessing it, that gives honeytokens the same power and advantages as traditional honeypots, but extend their capabilities beyond just physical computers [7].

A honeytoken is just like a honeypot, you put it out there and no one should interact with it. Any interaction with a honeytoken most likely represents unauthorized or malicious activity. What you use as a honeytoken, and how you use it, is up to you. Maintaining who is authorized to what can be complex, with false positives becoming a huge problem. Honeytokens can be used to solve and simplify this problem [7].

In this paper, we show that the emergence of SDN can reverse the trend and make the network easily configurable to build a robust defense against cleverly organized and targeted DDoS attacks. In particular, we show that the proposed scheme can effectively block the botnet-based DDoS attacks that do not exhibit any statistical anomalies that traditional network anomaly detection schemes can exploit for detection and blocking. The contributions of this paper can be summarized as follows:

- A DDoS detection and mitigation scheme that uses only the standard OpenFlow APIs is designed for operation in general SDN environments.
- The scheme is shown to work with minimal support from the server under attack. In particular, the server does not have to have a physical replicated service unlike the URL redirection schemes.
- The scheme is implemented as a SDN application that runs on a popular SDN controller, POX [9].
- A CAPTCHA mode is implemented on the server under attack such that service to legitimate users is not affected.
- The implemented code is tested through emulation on Mininet [10], where it is shown that DDoS attack using botnet is effectively blocked.

The rest of the paper is organized as follows. Section II gives details about the related work in the past. Section III defines the system architecture considered. It first clarifies the assumptions as to the behavior of the components in the SDN controlled network. Then it explains the main ideas, in terms of the functions and the interactions of the system components. Finally, it explains different modes of operation and their respective workflows. Section IV discusses how the designed system is implemented in Python to run on the POX [9] controller, along with the standard OpenFlow interfaces used in the implementation. This section validates the implemented Python logic using the Mininet [10] emulator. The emulation shows that the POX application successfully blocks the DDoS attack traffic besides maintaining service to legitimate traffic. Ultimately, section V concludes the paper with a consideration for future work.

II. RELATED WORK

Although security is said to be a "killer app" for the SDN, most existing security-related works in SDN is focused on the vulnerabilities of SDN and on how to resolve them and there is a dearth of work on using SDN to improve security in future networks. In the literature, there are a couple of prior works that take resemblance to our work. *Jafarian et al*. propose to mutate the server address frequently so that the network scanning prior to attack delivers an address of the victim that will be invalidated at the time of the attack [11]. But this scheme requires support from external network components such as DNS, and is not scalable. Braga et al. [12] discuss how to detect DDoS attacks in SDN. In the proposed scheme, the detection application on the NOX controller computes six features by obtaining flow statistics from switches through the OpenFlow interface A shortcoming of the scheme, however, is that it assumes IP address spoofing by the attacker. The latest is Lim et al [1] proposes a DDoS blocking scheme applicable to a SDN-managed network. The scheme requires communication between the DDoS blocking application running on the SDN and uses CAPTCHA for giving the redirection address received from the server. [2] Discusses a Click inspired programming framework that enables easy implementation of security applications through modular composition of security constructs

III. SYSTEM ARCHITECTURE

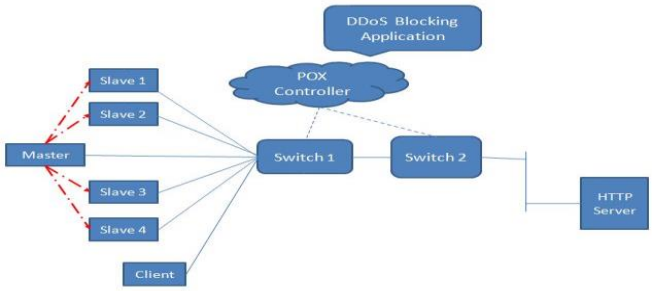

Fig. 1. System Architecture

Figure 1 shows the simplified illustration of the system architecture. It is composed of a protected service provided by the web server, SDN controller (POX) and the applications running on the controller, OpenFlow switches that are controlled by the controller through the OpenFlow interface. In addition, there is 1 legitimate client of the service, 4 bots and 1 botmaster. For subsequent discussions based on the system architecture, we make the following assumptions about the components in this paper.

- HTTP server behind the SDN-based network establishes no communication channels with the DDoS blocking application that runs on a SDN controller, e.g. POX. For convenience, this blocking application is called DDoS Blocking Application (DBA) in this paper.

- The DDoS attack is mounted by a botnet, and the botnet does not use IP address spoofing. The non-IP spoofing assumption contrasts to some related works, and is based on recent trend where most botnets do not have to rely on IP spoofing to confound any tracking effort.

Based on these assumptions and the system architecture depicted in Fig. 1, a SDN-based DDoS blocking scheme and in particular, the DDoS blocking application (DBA) component has been designed.

A. Work flow

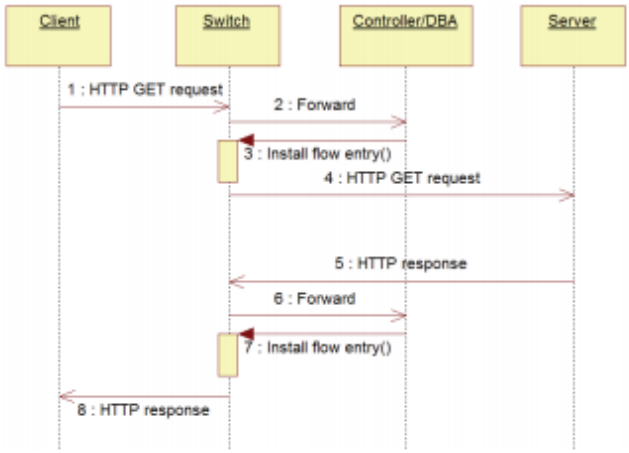

Fig. 2. New flow arrival (HTTP GET and response) [1]

When a new request from a client arrives at an OpenFlow switch, it does not match any existing flow. It is reported to the controller, which directs the switch to create a flow table entry for the new packet (Fig. 2). When the server responds to the request from the client, another flow entry is created at the switch, but in the reverse direction [1].
Using the new flow reports, DBA monitors the total number of flows at each flow switch. Meanwhile, the server monitors

metrics such as number of requests per interval and number of requests in last interval to indicate a possible DDoS attack. When the DBA determines that a DDoS attack has commenced and the server collapse is imminent, it starts putting the sources whose requests per interval exceed the nominal amount and ultimately dropping all the packets that are destined to the server. Any host address that is present in the blacklist maintained by the DBA is not able to get through to the server.

The server also keeps on monitoring the number of requests coming from the source per interval to determine whether the attack has been commenced. Once the server detects an attack, the server goes in the CAPTCHA mode wherein for any subsequent requests it asks users to authenticate themselves by entering the CAPTCHA. In this way the legitimate and illegitimate users are differentiated.

In this paper, CAPTCHA is a randomly generated string that has a mix of ASCII as well as numeric characters. The length of CAPTCHA is set to be eight. But any scheme that makes the bots experience difficulty in understanding the redirect message will serve the purpose.

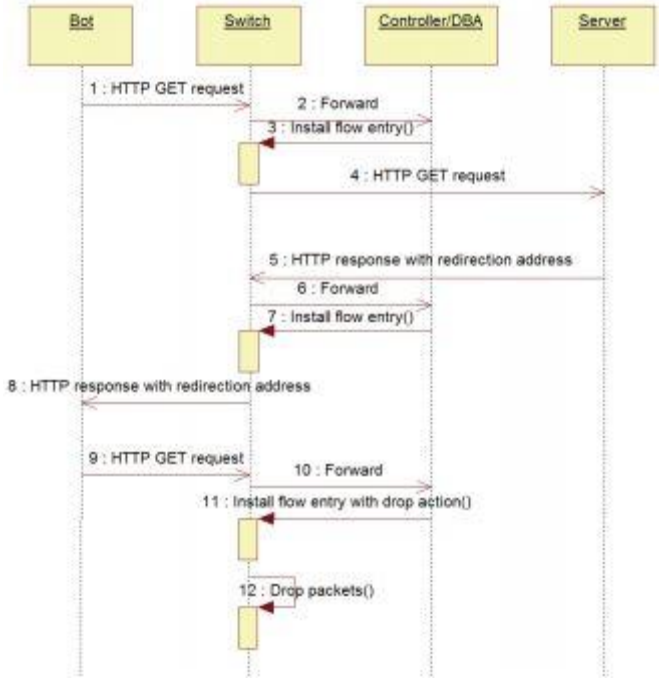

Fig. 3. System behavior against repeated bot requests [1].

When a client is classified as a bot, a flow entry with "drop" as the associated action is installed (Fig. 5, step 11). We will call such entry the Drop entry for convenience. One problem is the flow table explosion due to the explosive growth of the Drop entries that are created by bot requests [1]. Therefore, DBA instructs all flow switches downstream of the ingress switch to entirely delete the flow entry for the bot. As a consequence, Drop entries will remain only in the perimeter switches.

B. POX and OpenFlow interfaces

The workflow discussed above is implemented in POX, a Python-based SDN controller. The Python implementation of the entire DBA logic is approximately 200 lines. The code is immediately usable on the POX platform, but for a large-scale test it can be implemented on the Mininet emulator in the next section.

The following two standards OpenFlow APIs are enough to implement most of the logic required in the workflow that:

- OFPT_PACKET_IN
- OFPT_FLOW_MOD

OFPT_PACKET_IN message is used by the OpenFlow switch to report the arrival of a certain packet. Most typically, it takes place when the packet does not match any flow in the flow table.
OFPT_FLOW_MOD message is used in our system. It is used by the controller to instruct the OpenFlow switch to create, delete, or modify a flow table entry. In each case, the command part of the message is one of the following:

- OFPFC_ADD
- OFPFC_DELETE
- OFPFC_MODIFY

OFPFC_ADD is used when a flow table entry should be created at the flow switch. Obviously, it takes place when a new flow arrives. Another use of the command is when a source MAC address is found to be a bot by DBA, and the flow table entry to drop the packets from the bot should be installed at the flow switch. Normally, OFPFC_DELETE is used when a flow terminates. Additionally, in our system, it is used to purge all unnecessary flow entries after a source MAC address is determined to be a bot. By the time the event takes place, there entries created by the requests from the bot, and as many are entries created by the server issuing the redirection responses to them. OFPFC_MODIFY is not used in our system.

When a flow table entry is made, an action is associated with it. The following actions are used in the system, both of which are mandatory in the OpenFlow standard:

- Output
- Drop

Output is the action to forward the packet. Drop is obviously to drop the given packet. It is used after the flow entry to match and drop the bot-originated packets that are installed. In order to find a matching flow in the OpenFlow API, various fields can be used in the packet, such as IP protocol number, IPv4 addresses, IPv6 addresses, TCP/UDP port numbers, incoming switch port, Ether type, and MAC addresses [1]. In this project, the parameters source and destination MAC

addresses and the source port number are used to match packet entries.

IV. EXPERIMENTS

In order to validate the correctness of the logic in the proposed scheme and evaluate its performance, we test the DBA code for POX. For a massive attack experiment, we choose to use network emulator called Mininet [10]. Mininet is a network emulator that can model a network of virtual hosts, switches, controllers, and links for SDN. Compared to simulators, Mininet runs real, unmodified code including application code, OS kernel code, and control plane code (both OpenFlow controller code and Open vSwitch code), and easily connects to real networks. In particular, our DBA code that runs on real POX [9] platform also runs on Mininet in emulation. In emulation, Mininet hosts run standard Linux network software, and Mininet switches support OpenFlow [6] for highly flexible custom routing and software defined networking. For our experiments, we use Mininet 2.1.0, which supports OpenFlow v1.3 standard. Also on the Mininet testbed, we create a large number of bots to mount a DDoS attack on a protected server in a SDN-managed network.

A. Parameters

Table 1: Mininet emulation parameters

parameter	Meaning	Default value
θ	Flow count threshold	40
C_{total}	Total No. of possible connections	100
C_{avg}	No. of average connections per interval	0
C_{last}	No. of connections in last interval	0
n	No. of legitimate users	1
k	No. of bots	4
m	No. of master bots	1
-	Server refresh rate	2 secs
-	Request arrival process	Poisson

Table 1 summarizes the parameters used in our experiment. We assume there are two groups of clients, legitimate and malicious. Namely, bots are modeled as being more active than legitimate users, but not as anomalously as can be noticed by the network. The requests are issued with the exponential inter-arrival time distribution. Today's botnets can be huge, sometimes boasting up to half a million-strong army of bots. Unfortunately, we could not emulate such a large-scale attack, due to the Mininet limitation. As Mininet can emulate only up to certain limited number of nodes we scaled down the server to have a maximum of C_{total} connections at a time. The server and the DBA know when a large fraction of the maximum allowable connections is exceeded. In this paper, we assume that the server suspects a DDoS attack if the load reaches 50% of the maximum capacity. We assume that there are n = 1 legitimate users and k = 4 bots that issue requests at their given intensities. For each request, the server is assumed to be responsive, so the response is returned in no time.

B. Emulation setup

- Honeytokens:
  In this project honeytokens are used as a means to lure malicious users. And hence the web server has two files namely, 'asdf.txt' which is a legitimate file and 'classified.txt' is a honeytoken. The server maintains its own blacklist to track the IP of those users that try to access the honeytoken.

- Attack Mode:
  In attack mode, it is assumed that there is no DBA type application running over the controller side and that it functions just as a normal layer 2 learning switch. Hence, whenever the SDN controller (POX) is switched ON, typical flooding action is observed. The master (m) waits for all the slaves (bots (k)) to connect to it and once the last bot connects, it starts issuing commands to bots to start the flooding attack.

- Defense Mode:
  In defense mode, DDoS blocking Application (DBA) runs over POX, which is used for both detection and mitigation of the attack. DBA keeps on monitoring number of flows based on (src MAC, dst MAC) pair and once the flow count > threshold (θ), the corresponding source and destination MAC addresses are added to the Blacklist and respective packets are dropped. Furthermore, the server in this not a typical web server, it keeps on monitoring C_{avg} and C_{last}.
  If $C_{last} > C_{avg}$ CAPTCHA mode is enabled. Hence, for any subsequent request the server will respond with a random string for authentication and if sent. (CAPTCHA) == received. (CAPTCHA) user is authenticated and only then the client will receive its response.

C. Emulation results

In attack mode, since there is no blocking application running over the SDN controller, the controller instructs switches to add all the entries into its flow table and forward them. Hence, the switches forward all the GET request from the bot towards the server; as a result the server is overloaded and unable to provide any further service resulting in Distributed Denial of Service (DDoS). Therefore, the server shuts down for prevention of any data loss and has to reboot.

However, in the defense mode there is attack detection and mitigation application running over POX that keeps track of flows and once it detects the attack it instructs switches to

drop the particular flows there by reducing the number of requests reaching to the server. Additionally, the DBA and server are so finely calibrated in a way that once DBA detects an attack the serve goes into the CAPTCHA mode for differentiating legitimate user from the bot and ultimately preventing illegal access to the data.

Moreover, in the case of honeytokens the probability of an attack from a blacklisted IP is more therefore, all the IP addresses in the blacklist are denied access to the actual data and the administrator can put these users in the controller's blacklist thereby preventing any future attacks or malicious activity.

Finally, from the observations the system provides overall functionality in an integrated manner wherein attacks are precisely detected and mitigated, controlled access to the important data is ensured and prevention of future attacks is also guaranteed.

D. Discussion

As there are a variety of CAPTCHA techniques and as many CAPTCHA breakers, the proposed scheme cannot entirely depend upon the strength of CAPTCHA to deter bots. It is up to the server to present the redirection information to legitimate clients in a format machine decoding is computationally expensive. The drawback of [1] was that the scheme required communication between the DDoS blocking application running on the SDN controller and the server to be protected. But, our approach reduces the dependence on the pre-arranged cooperation between the SDN controller and the protected server so that SDN can provide transparent protection to servers inside it.

V. CONCLUSION

In this project, a DDoS detection and mitigation scheme is proposed that is applicable in a SDN-managed network. The scheme does not require communication between the DDoS blocking application running on the SDN controller and the server to be protected. Controlled access to the important data is ensured using CAPTCHA. Furthermore, honeytokens are used as a mechanism for early detection and confirmation of advanced insider threats. The proposed scheme culminates in an application that orchestrates the defense through simple and few OpenFlow interfaces. The implemented code for POX controller is validated on Mininet emulator, and is shown to block DDoS attack from bots.

REFERENCES

[1] Lim, S.; Ha, J.; Kim, H.; Kim, Y.; Yang, S., "A SDN-oriented DDoS blocking scheme for botnet-based attacks," Ubiquitous and Future Networks (ICUFN), 2014 Sixth International Conf on , vol., no., pp.63,68, 8-11 July 2014.

[2] S. Shin et al., "FRESCO: Modular Composable Security Services for Software-Defined Networks," Proc. ISOC NDSS, 2013.

[3] Open Networking Foundation, OpenFlow Switch Specification v.1.4.0, Oct. 2013.

[4] Kreutz, D.; Ramos, F.M.V.; Esteves Verissimo, P.; Esteve Rothenberg, C.; Azodolmolky, S.; Uhlig, S., "Software-Defined Networking: A Comprehensive Survey," Proceedings of the IEEE , vol.103, no.1, pp.14,76, Jan. 2015

[5] Spitzner, L., "Honeypots: catching the insider threat," Computer Security Applications Conference, 2003. Proceedings. 19th Annual , vol., no., pp.170,179, 8-12 Dec. 2003

[6] Open Networking Foundation, "Operator network monetization through OpenFlow-enabled SDN," Apr. 2013.

[7] Ramneek, Puri (2003-08-08). "Bots & Botnet: An Overview" (PDF). SANS Institute. Retrieved 12 November 2013.

[8] Honeytokens: The Other Honeypot Lance Spitzner 2003-07-17

[9] About POX, http://www.noxrepo.org/pox/about-pox/.

[10] Mininet. http://mininet.org/overview.

[11] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "OpenFlow Random Host Mutation: Transparent Moving Target Defense using Software Defined Networking," Proc. ACM HotSDN, pp. 127-132, 2012.

[12] R. Braga, E. mota, and A. Passito, "Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow," Proc. IEEE LCN, pp. 408-415, 2010.