

LAMP: Prompt Layer 7 Attack Mitigation With Programmable Data Planes

Garegin Grigoryan
gg5996@rit.edu
Rochester Institute of Technology*

Yaoqing Liu
liu@clarkson.edu
Clarkson University

Abstract—While there are various methods to detect application layer attacks or intrusion attempts on an individual end host, it is not efficient to provide all end hosts in the network with heavy-duty defense systems or software firewalls. In this work, we leverage a new concept of programmable data planes, to directly react on alerts raised by a victim and prevent further attacks on the whole network by blocking the attack at the network edge. We call our design LAMP, Layer 7 Attack Mitigation with Programmable data planes. We implemented LAMP using the P4 data plane programming language and evaluated its effectiveness and efficiency in the Behavioral Model (bmv2) environment.

I. INTRODUCTION

Layer 7 attacks target the resources of the application layer, such as web and database servers. A victim of Layer 7 attack might be a data center, a cloud service or a private network. There are multiple scenarios where Layer 7 attacks can disrupt the services provided by a network. For example, a DDoS attack known as HTTP flood exhausts web servers and databases of a network by sending a great amount of POST or GET requests. The attack can be reinforced, when it is conducted by large botnets, i.e., a network of compromised devices controlled by an attacker. IoT (Internet of Things) devices are the most attractive victims to be lured into botnets, because of various security flaws. In particular, most often the users of IoT devices do not change the default login and password embedded by the manufacturer, which makes the IoT devices vulnerable to dictionary attacks [1]. Moreover, IoT devices possess limited CPU resources for identifying a malicious user or an intrusion attempt [2].

The number of attacks at the application layer is growing, according to the "Global DDoS Threat Landscape Report" [3]. Current detection techniques for application layer attacks include wide range of measures including (1) Pattern analysis of HTTP requests; (2) Browsing behavior analysis using web logs; (3) Geo-location analysis of web clients; (4) Machine learning pre-profiling legitimate traffic; (5) Application layer challenges, such as as CAPTCHA; (6) JavaScript engine authentication and many others ([4], [5], [6], [7], [8]). All of these techniques require application data analysis with high computational capabilities, which are usually available at the powerful end hosts. Upon detection, the malicious traffic can be dropped based on their source IP addresses or other attributes, such as TCP/UDP ports or HTTP request information.

Normally, there are two ways to achieve this goal with the help of a victim end host. The first one is individual or local defense, where the machine running the application installs certain firewall/IDS rules to block the attack traffic. One of the main drawbacks of this approach is that the identified attacking information cannot be re-used by other end hosts in the same network. The second approach is cooperative or global defense, where a Software Defined Networking (SDN) controller can be used to collect the detection results and to install the corresponding OpenFlow-like rules to SDN-enabled switches. However, in this case, an SDN controller introduces additional complexity and overhead to the network operations. In addition, security of the SDN controller itself is another concern.

To enable prompt, cooperative, and efficient mitigation of Layer 7 attacks, in this work, we introduce a new approach by leveraging the Protocol Independent Switch Architecture (PISA) [9]. PISA allows us to program data planes directly without involving a centralized controller by using parser engines, match-action tables, ingress and egress pipelines in P4 language [9]. More specifically, we design LAMP, Layer 7 Attack Mitigation with Programmable data planes. In LAMP, we track the path of each flow coming into the victim network. If an end host application detects an intrusion attempt, it generates an attack alert by embedding a signal flag and the attacker's IP address in the IP option field of the reply packet. We assume that such application has privileges to modify IP packets' fields of the alert message. The alert is sent to the closest switch directly, which eventually forwards the packet to the ingress switch that carried the original malicious traffic. Upon receiving the alert, the ingress switch modifies its flow control policy to block the subsequent traffic from the attacker. Our new mitigation strategy yields at least three advantages: (1) It enables network-wide cooperative detection and mitigation of attacks. The detection results obtained by one end host can be re-used to benefit services for the entire network; (2) The volume of in-network malicious traffic is considerably reduced, since the network edge quickly blocks them; and (3) It enables lightweight and efficient network operations compared to existing SDN approaches, where an SDN controller is required to bridge the gap between application and network layer services. In addition, SDN incurs many additional messages, necessary to establish connections between the SDN controller and end hosts. Overall, we made the following contributions in this work:

*This work was done when Garegin Grigoryan was a student at Clarkson University.

(1) We designed LAMP, a new cooperative framework for prompt and efficient mitigation of Layer 7 attacks without the involvement of a centralized SDN controller;

(2) To the best of our knowledge, it is the first time that we designed a Layer 7 intrusion mitigation solution by leveraging the new concept of programmable data planes;

(3) We implemented LAMP in P4 [9], the language for programming the data plane of a switch. LAMP can re-use the same P4 code for all the switches in a network for the mitigation tasks.

(4) We emulated LAMP in the *bmv2* model [10], a virtual environment designed for testing P4-programmed switches. Our comparison with a similar SDN architecture shows that LAMP bears much less operational complexity and minimizes the number of malicious application messages reaching end hosts once an attack is detected.

II. RELATED WORK

In [15], Norman *et al.* studied application layer protocols used in modern IoT devices and their vulnerabilities. De Donno *et al.* in [2] presented a deep survey on DDoS attacks against the IoT. Particularly, the authors analyzed Mirai, an attack that was conducted using large number of IoT devices, compromised with application layer dictionary attacks and lured into the botnets. In [5], [7], [8], [6], the authors proposed various Layer 7 attack detection techniques, such as additional client tests (CAPTCHA, passwords, puzzles, JavaScript authentication); web logs analysis and building a profile of a legitimate user; analysis of the visiting history of clients; traffic classification. LAMP is compatible with all of these approaches since it is designed for fast mitigation of attacks after they are detected by a member(s) of the network.

Giotis *et al.* in [16] presented an SDN architecture for flow-based anomaly detection and installation of the mitigation rules on the edge switches. The mitigation rules are dropping the packets based on their source and destination IP addresses. Lim *et al.* [17] design DDoS blocking solution that changes the IP address of the victim and redirects the legitimate connections, in order to mitigate large botnet attacks. In our work, we present LAMP architecture, that blocks malicious traffic at the edge switches using programmable data planes.

III. DESIGN

We demonstrate an example of Layer 7 attack mitigation with LAMP on Figure 1. The attacker starts scanning resources of the victim network in Figure 1(b), e.g., a random dictionary attack in order to login into the most vulnerable end hosts. Since the attack is conducted against the application layer, the network layer devices, such as switches and routers, are unable to detect it, and they allow the packets to reach the end hosts. To track the entrance point of each external flow, *Switch 1* encapsulates its ID into the option field of the incoming IP packet's headers. The option is given a special type *INGRESS_SWITCH_INFO* to differentiate it from other possible options. Later, the switch ID and its option are dumped as the last switch on the path forwards the packet

to the final destination. Meanwhile, that switch records the mapping between the flow's source IP address and the edge (ingress) switch's ID.

In our scenario, the end-host *Server 2* detects the scanning attempt and sends the attack alert message back into the network (Figure 1(c)). The alert is encapsulated inside *ATTACK_ALERT* option. The attack alert contains the IP source address of the attacker. The switch (*Switch 3*) that receives the alert finds the corresponding ingress switch (*Switch 1*), adds the switch ID to the IP option, changes its type to *FORWARD* and sends the packet to the next hop towards that switch (*Switch 1*). Lastly, *Switch 1* installs necessary entries to drop the packets from the attacker before they enter the network. To implement LAMP using the P4 language, we need to program the following components over a programmable data plane: (1) The parser; (2) Match-action tables; (3) Ingress and egress flows of a switch. In the rest of this section, we give the detailed description of our modifications in each of those components.

A. Parser

To implement the parser, we first define (1) The headers that should be decoupled and read from each packet; (2) The possible option numbers used in LAMP. We create 3 types of options with the following option numbers from the unassigned range: *ATTACK_ALERT* = 31; *INGRESS_SWITCH_INFO* = 29; *FORWARD* = 27.

In addition to Ethernet (*ethernet_t*), IP (*ipv4_t*) and the standard IP option header (*ipv4_option_t*), we defined the switch header (*switch_t*) and the alert header (*block_t*), that shall contain the edge switch's ID and attacker's IP address respectively. *switch_t* and *block_t* are placed within the payload of *ipv4_option_t*, so LAMP's parser does not need to decouple the Layer 4 (the Transport Layer) headers. In the meantime, for the internal packets of a network, the parser will not decouple *switch_t* or *block_t* headers, because those packets shall not contain any of the above-mentioned IP options. Thus, we exclude the additional overhead for the internal traffic, which in some cases constitutes more than 75% of the total traffic within a network [11]. LAMP's parser's state transitions are illustrated on Figure 2.

B. Match-action tables

In LAMP, match-action tables are used for (1) Forwarding the packets based on their IP destination address; (2) Adding the edge switch's ID to a packet that comes from outside the network; (3) Removing the edge switch's ID from a packet whose next hop is an end host; (4) Forwarding the attack alert based on the edge switch's ID. While the tables' structure and actions are defined in the data plane, the content of the tables is regulated by the control plane. We assume that the centralized or a distributed control plane can automatically populate match-action tables using API generated by P4 compiler.

(1) *ipv4_lpm*: As in the "Simple Switch" presented by P4 designers [12], table *ipv4_lpm* contains the prefixes and the corresponding next hops (output port and MAC address

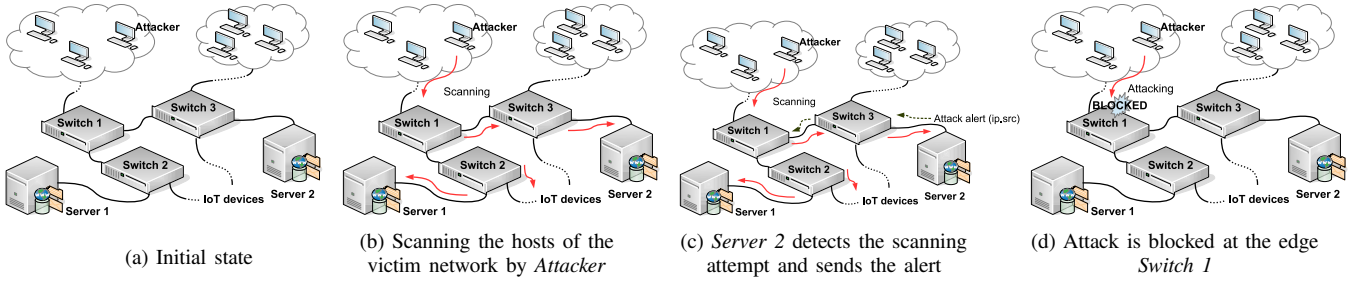


Fig. 1: Scenario of Layer 7 attack mitigation with LAMP

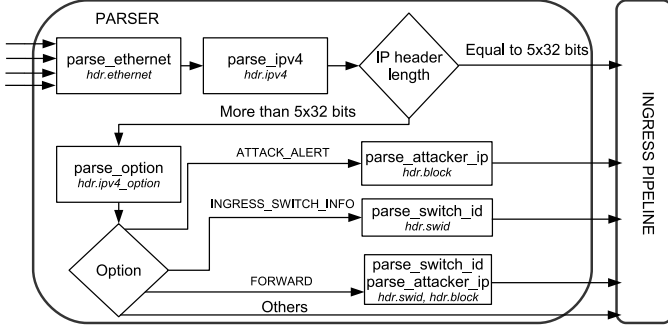


Fig. 2: Parser in LAMP

information). The destination IP address of a packet will be matched against one of those prefixes in order to get the next hop. Based on the match, the packet is either forwarded (action *ipv4_forward*) or dropped.

(2) *swid_add*: Contains the mapping of the current switch's ID and the list of ports that connect the switch to an external network. On receiving a packet from one of those ports, the switch will attach its ID to the packet (action *add_swid*). In addition, the packet will be marked as "to be checked" using the packet's metadata field (*meta.check_source_ip=1*). The table is empty for internal switches.

(3) *swid_remove*: Contains ports that connect to end hosts and the option numbers we specified for Layer 7 attack mitigation. LAMP needs to remove our previously attached *INGRESS_SWITCH_INFO* option via *remove_swid* action from the packets that are reaching an end host at their next hop. Meanwhile, the switch needs to store the mapping between the attached ingress switch ID and the hashed value of the packet's source IP address. The table should be empty for switches that are not directly connected to end hosts.

(4) *swid_forward*: Contains the list of destination switch IDs, the next hop ports and the corresponding MAC addresses in the network. This table may incur two different actions based on the value of packet's destination switch ID *hdr.switchID.swid*: (a) If the value equals to the current switch ID, run the action *block*, that will install a drop entry into the blacklist hash table for the attached source IP address in the alert message; (b) Otherwise, run the action *ipv4_forward*, that will forward the packet to the next hop towards the destination switch whose ID equals to *hdr.switchID.swid*. To enable such workflow, the control plane needs to correctly initialize *swid_forward* table.

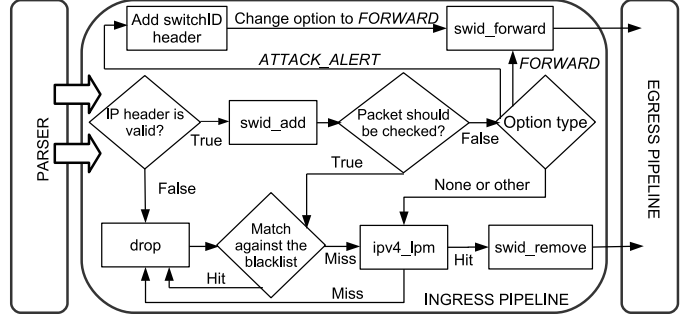


Fig. 3: Ingress pipeline in LAMP

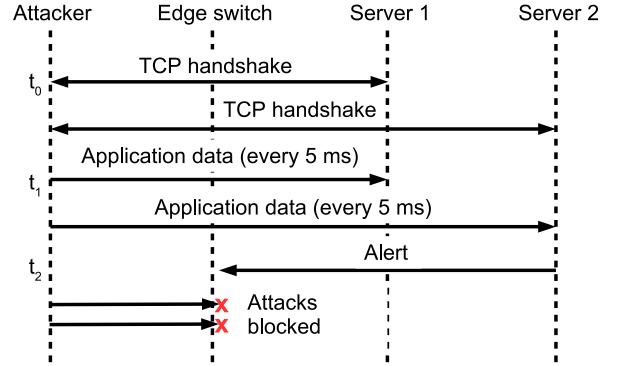


Fig. 4: Evaluation scenario

C. Control flow

1) *Ingress flow*: After a packet is parsed (Figure 2), its header fields are transmitted to the ingress pipeline as shown in Figure 3. At first, LAMP checks the validity of the IP header. If valid, the packet's input port is matched against *swid_add* table. Two cases may happen: (1) If the port faces an external network, the current switch ID is attached to the header option of the packet. In addition, the packet's source IP address will be matched against the blacklist. If a match is found, it indicates that this packet was from an attacker and LAMP drops it; (2) Otherwise, the packet's IP option is checked. If there is an *ATTACK_ALERT* option, the switch first finds the corresponding ingress switch and adds its ID to the packet header option. Then it changes the option to *FORWARD* and forwards the packet towards that ingress switch. In case the packet does not have above-mentioned options, it is matched against *ipv4_lpm* table to obtain the next hop port and MAC address. Lastly, the packet is matched against *swid_remove* table to remove option *INGRESS_SWITCH_INFO* if it exists and the next hop for the packet is an end host.

Architecture	LAMP	SDN
Measurement		
Total	278	1288
Maximum	10	106
Minimum	5	5
Average	9	43

TABLE I: The number of invalid HTTP requests that *Server 1* received in SDN and LAMP architectures

2) *Egress flow*: For the egress flow, LAMP simply deparses the packet header in the following order: *hdr.ethernet*, *hdr.ipv4*, *hdr.ipv4_option*, *hdr.block*, *hdr.switchID*. P4 program automatically checks if each of these headers is valid and omits the header if not. We omit P4 code listings ingress control flow and the deparser due to the space constraints.

IV. EVALUATION

We emulated LAMP in the Behavioral Model (bmv2) [10], which provides a P4 software switch with the compiler using Mininet virtual environment [13]. We compared LAMP with a similar architecture, implemented using SDN and OpenFlow [14] in Mininet. The topology for the experiment is similar to one illustrated on Figure 1. We used the following scenario for the experiment (see Figure 4): At the moment t_0 , the *Attacker* establishes TCP connection with *Server 1* and *Server 2*; at t_1 , it starts sending packets with invalid HTTP requests with a rate 200 packets/s. As soon as *Server 2* detects the attack, it sends an alert message into the network (t_2). In case of LAMP, the message reaches the edge switch, where a blocking entry is installed into the blacklist. In case of SDN, the message is captured by an SDN controller, which figures out the corresponding ingress switch and installs a drop rule into its OpenFlow table. In our experiment, we emulated 30 attacks for both LAMP and SDN architectures. We intended to observe how many messages can go through the network to reach the other victim before the attack can be blocked. Interestingly, unlike LAMP, the SDN emulation produced fluctuating results, as it can be seen in Table I. Overall, in LAMP, *Server 1* received only 278 invalid HTTP requests, 1010 packets less than that received by *Server 1* in the SDN architecture. Moreover, the maximum number of such requests during a single attack in LAMP is 10, while in SDN it reached 106. But, in some cases, both LAMP and SDN controller acted fast enough to block *Attacker*, so *Server 1* received only 5 invalid HTTP requests. On average, in SDN, *Server 1* received 80% more invalid HTTP requests than that in LAMP. We attribute it to the use of a centralized controller that introduces additional overhead and complexity. In LAMP, attack alerts are processed fully in the data plane, which makes attack mitigation significantly faster. In the meantime, other factors might have affected the performance of two emulated architectures, such as the differences and deficiencies of P4 and SDN implementations in Mininet.

V. CONCLUSION

In this work, we presented LAMP, an architecture for Layer 7 attack mitigation with programmable data planes. To the best of our knowledge, for the first time we leveraged

Protocol Independent Switch Architecture (PISA) to design a cooperative mitigation solution against the application layer attacks. We presented the detailed solution of the new mitigation architecture in LAMP, including the modified parser, match-action tables and the ingress flow. We implemented LAMP in the P4 language and emulated it in the Mininet virtual environment. Compared to a similar Software Defined Networking architecture, LAMP mitigates the Layer 7 attacks more quickly and minimizes the number of malicious application layer messages that are sent to victims of the same network.

REFERENCES

- [1] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zuolkernan, "Internet of things (iot) security: Current status, challenges and prospective measures," in *Internet Technology and Secured Transactions (ICITST), 2015 10th International Conference for*. IEEE, 2015, pp. 336–341.
- [2] A. Spognardi, M. De Donno, N. Dragoni, and A. Giaretta, "Analysis of ddos-capable iot malwares," *Annals of Computer Science and Information Systems*, vol. 11, pp. 807–816, 2017.
- [3] Imperva Incapsula, "Q1 2017 Global DDoS Threat Landscape Report," 2017. [Online]. Available: <https://www.incapsula.com/blog/q1-2017-global-ddos-threat-landscape-report.html/>
- [4] Infosec Institute, "Layer 7 DDoS Attacks: Detection And Mitigation," 2013. [Online]. Available: <http://resources.infosecinstitute.com/layer-7-ddos-attacks-detection-mitigation/>
- [5] J. D. Ndibwile, A. Govardhan, K. Okada, and Y. Kadobayashi, "Web server protection against application layer ddos attacks using machine learning and traffic authentication," in *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, vol. 3. IEEE, 2015, pp. 261–267.
- [6] R. Bronte, H. Shahriar, and H. M. Haddad, "Mitigating distributed denial of service attacks at the application layer," in *Proceedings of the Symposium on Applied Computing*. ACM, 2017, pp. 693–696.
- [7] S. Prabha and R. Anitha, "Mitigation of application traffic ddos attacks with trust and am based hmm models," *International Journal of Computer Applications IJCA*, vol. 6, no. 9, pp. 26–34, 2010.
- [8] C. Wang, T. N. Miu, X. Luo, and J. Wang, "Skyshield: A sketch-based defense system against application layer ddos attacks," *IEEE Transactions on Information Forensics and Security*, 2017.
- [9] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.
- [10] "Behavioral model repository," 2017. [Online]. Available: <https://github.com/p4lang/behavioral-model/>
- [11] Timothy Prickett Morgan, "Cisco: Data Center Traffic To Quadruple Thanks To Clouds," 2012. [Online]. Available: <https://www.itjungle.com/2012/10/29/tfh102912-story06/>
- [12] Barefoot, "Simple switch," 2012. [Online]. Available: https://github.com/p4lang/tutorials/blob/master/P4D2_2017_Spring/exercises/ipv4_forw
- [13] K. Kaur, J. Singh, and N. S. Ghuman, "Mininet as software defined networking testing platform," in *International Conference on Communication, Computing & Systems (ICCCS)*, 2014, pp. 139–42.
- [14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.
- [15] J. Norman and P. Joseph, "Security in Application Layer Protocols of IoT: Threats and Attacks," in *Security Breaches and Threat Prevention in the Internet of Things*. IGI Global, 2017, pp. 76–95.
- [16] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments," *Computer Networks*, vol. 62, pp. 122–136, 2014.
- [17] S. Lim, J. Ha, H. Kim, Y. Kim, and S. Yang, "A SDN-oriented DDoS blocking scheme for botnet-based attacks," in *Ubiquitous and Future Networks (ICUFN), 2014 Sixth International Conf on*. IEEE, 2014, pp. 63–68.