Dynamic Security Orchestration for CDN Edge-Servers

Elaheh Jalalpour*, Milad Ghaznavi*, Daniel Migault†,
Stere Preda†, Makan Pourzandi†, Raouf Boutaba*

*University of Waterloo, Waterloo, ON, Canada †Ericsson Research, Montreal, QC, Canada
*{ejalalpo | eghaznav | rboutaba}@uwaterloo.ca,
†{daniel.migault | stere.preda | makan.pourzandi}@ericsson.com

***Abstract*—Content Delivery Networks (CDNs) aim to provide high Quality of Service (QoS) in serving digital content. To achieve high QoS, CDNs employ edge-servers that cache content in the vicinity of end-users. Edge-servers are vulnerable to attacks that degrade the QoS of end-users. Protecting edge-servers against these threats is vital and complex. The attack mitigation must be immediate, and its overhead should have the least impact on the QoS of legitimate end-users.**

In this paper, we demonstrate a software-based security system that can be programmed to automatically react to threats by deploying and managing security function chains. Using high-level security policies, a network operator can program a desired system behavior. We demonstrate how our system automatically deploys security function chains to handle real-world threats.

I. Introduction

Content Delivery Networks (CDNs) are crucial in delivering digital content to end-users. To provide high Quality of Service (QoS), CDNs cache content in *edge-servers* placed at points of presence in the vicinity of end-users. Attacks against edge-servers disrupt content delivery services and cause QoS degradation which may result in revenue and reputation loss for CDN providers.

To lessen possible attack damages, security services must react fast. Because of their processing overhead, security services might adversely affect the QoS of legitimate end-users. In reacting to threats, security services must be deployed *dynamically* and *automatically* to process suspicious traffic flows and without negatively affecting legitimate end-users. Furthermore, attacks are becoming more sophisticated, and new attacks are launched on a daily basis, thus protection systems should be adaptive in their ability to handle new threats and integrate new security services.

Traditional security mechanisms do not satisfy all the above requirements. Defense using *hardware* security functions are constrained to the resources and functionality embedded in hardware. Redirecting traffic to *scrubbing-centers* for inspection adds latency and affects QoS. By leveraging *software defined networking*, *network function virtualization*, and *service function chaining*, virtualized security functions can be dynamically instantiated, modified, scaled, and released on-demand. Such flexibility in security orchestration facilitates the development of protection system that effectively secure edge-servers. However, current security solutions leveraging *softwarization* are not tailored to the security requirements of a CDN's edge-servers. Some solutions (e.g., VFence [8], Bohatei [7], and Holistic DDoS defense [5]) focus on only DDoS attacks and provide *static* DDoS mitigation mechanisms, i.e., they require a network operator to manually configure and provision mitigation mechanisms. Other solutions require substantial changes in existing infrastructures [13], [10].

978-1-5386-4633-5/18/$31.00 © 2018 IEEE

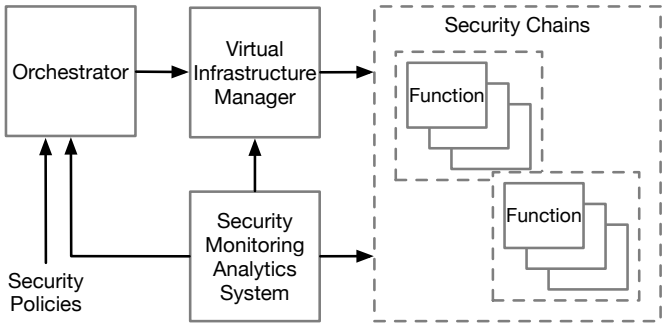


Fig. 1: Architecture from [9]

In this paper, we demonstrate a security system [9] that automatically and dynamically deploys security function chains in *response* to threats. The operator specifies high-level *security policies* translated by our system into mitigation actions by creating, modifying, or deleting security functions chains. We present our system in Section II and demonstrate its capabilities in Section III.

II. System Design and Implementation

Our system protects a *virtual edge-server* composed of a number of physical or virtual servers. It instantiates and reconfigures security chains in *response* to threats and a virtual edge-server's dynamicity. This *reactive* behavior is governed by *security policies* specified by the operator. As shown in Figure 1, our system is composed of three components:

1) Orchestrator: Governed by security policies, the orchestrator interacts with other system components to receive security alerts and to deploy security chains. We adapt $\mathcal{L}_{active}$ [6], a language following *event-condition-action* paradigm, for the specification of security policies. A security policy specifies if a specific event happens, and if particular conditions hold, the

system must execute a certain sequence of actions. Essentially, the enforcement of these policies translate into the deployment, modification, and removal of security chains.

2) Virtual Infrastructure Manager (VIM): This component manages security function chains and the virtual edge-server's resources. It provides a northbound API to create and delete a chain, insert and delete a function to and from a chain, and query information about deployed chains. We leverage Docker [11], network service header [12], and open virtual switch [2] in the VIM implementation.

3) Security Monitoring Analytics System (SMAS): This component collects data about security chains using the VIM's API and monitors a virtual edge-server's resources. SMAS analyzes these data and sends alerts to the orchestrator which may trigger security actions. In our current implementation, SMAS monitors and analyzes network-bandwidth, storage, memory, and processing resources.

III. Demonstration

CDNs commonly use *rate-limiting* in response to network and application layer attacks [3], [4]. There are different rate-limiting mechanisms, for example rate-limiting in different layers of the protocol stack and per content, end-user, server, and geography. In the following two demonstration scenarios, we present how our system mitigates network layer and application layer threats using rate-limiting.

A. Network Layer Rate Limiting

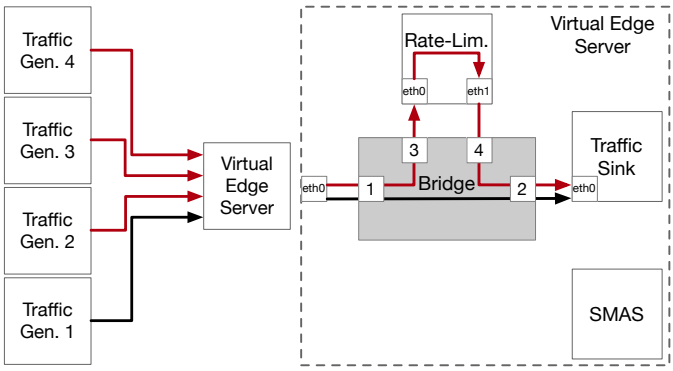
Context. Network flooding attacks (e.g., TCP flooding and SYN flooding attacks) exhaust the resources of an edge-server and make the service unavailable to legitimate end-users. In a multi-stage scenario, we program our system to rate-limit traffic *per-IP* which is a common rate-limiting mechanism to mitigate TCP flooding attacks.

Environment Setup. We employ a cluster of machines each of which equipped with 16 GB RAM, 8-cores 3.30 GHz Xeon CPU, and 10 Gbps NIC. As the device under test, a server hosts security chains and an active daemon of our system. One to four servers generate traffic load using `iperf client`, and a server running `iperf server` acts as the traffic sink.

Overview. Figure 2a illustrates the experimental setup of this demo. We use Traffic Gen. 1 to send legitimate traffic and Traffic Gen. 2 to 4 to generate flooding traffic. Our system is programmed using security policies[1] presented in Figure 3 to setup the mitigation security chain shown in Figure 2b. Table I explains the details of the stages of this demonstration scenario.

Details. The demo begins with sending only legitimate traffic from Traffic Gen. 1 (stage 1). We incrementally increase flooding traffic using Traffic Gen. 2 to 4 that results in a throughput drop of the legitimate traffic (stages 2 to 4). If the number of connections surpasses a predefined threshold, too_many_con alert is raised to the orchestrator. As shown in Figure 3, upon receiving the too_many_con alert, the orchestrator is instructed by Policies 1-3 to deploy chain r containing a $Rate\text{-}limit$, if no rate-limiting service exists. As shown in Figure 2b, this chain applies per-IP rate-limiting on traffic coming from Traffic Gen. 2 to 4, while the legitimate traffic coming from Traffic Gen. 1 is exempted and directly served. In this way, the flooding traffic is rate-limited, and the throughput of the legitimate traffic is immediately recovered (stage 5).

[1]For more details about the policies, we refer readers to our paper [9].



(a) Demo Setup (b) Mitigation

Fig. 2: Network Layer Rate Limiting

TABLE I: The Stages of Responsiveness Experiment

Stage	Flooding traffic share	Active traffic generators
1	0%	Traffic Gen. 1
2	50%	Traffic Gen. 1 and 2
3	66.6%	Traffic Gen. 1, 2, and 3
4	75%	Traffic Gen. 1, 2, 3, and 4
5	Limited to 1 Gpbs	Traffic Gen. 1, 2, 3, and 4

$$
\begin{aligned}
&too\_many\_con \textbf{ initiates } create\_chain(r: \\
&\quad \text{<“not src net 129.97.124.0/24”}, 1, 2\text{>}, \\
&\quad \{f\text{:}Rate\text{-}Lim.\}) \\
&\quad \textbf{if not } chain(r) \qquad (1) \\
&lim \textbf{ after } create\_chain(r) \\
&\quad \textbf{if true} \qquad (2) \\
&lim \textbf{ initiates } run(f, \text{“}rate\_limit.sh\text{”}) \\
&\quad \textbf{if true} \qquad (3)
\end{aligned}
$$

Fig. 3: Network Layer Rate Limiting Policies

B. Application Layer Rate Limiting

Context. An important CDN application is Video on Demand (VoD) streaming. CDNs use HTTP-based media streaming, such as HTTP Live Streaming (HLS) and HTTP Smooth Streaming (HSS) to provide VoD services. These protocols enable end-users to request different media qualities in near-real-time. An original media is encoded and segmented into multiple chunks with different bit-rates and formats. These chunks are listed in a *manifest* file. A common VoD session

starts with an end-user acquiring a manifest file and then issuing subsequent requests of individual chunks. Abusive end-users attempt application-layer attacks by requesting video chunks repeatedly. In response, CDNs need to mitigate such threats by rejecting requests with abnormal rates. In this demo, we present how our system can mitigate these threats.

Environment Setup. We use the same cluster of servers employed in the first demo. A server runs an `Apache` based streaming engine and an active daemon of our system. This server maintains video chunks and their corresponding manifest files and uses the HLS protocol to serve VoD requests. End-users use `VLC` to request the manifest files. To implement a legitimate behavior, we use `VLC` to issue requests for video chunks automatically. On the other hand, an abusive behavior is implemented by issuing frequent requests for one or several video chunks. We use `curl` to request chunk URLs according to the manifest files and retrieve the video chunks more frequently compared to `VLC`.

Overview. Figure 4a depicts the setup of this demo. We use End-user 1 to send legitimate requests, and End-user 2 and 3 to generate abusive requests. Figure 4b illustrates the mitigation setup, and Figure 5 presents the policies that are used to program the orchestrator.

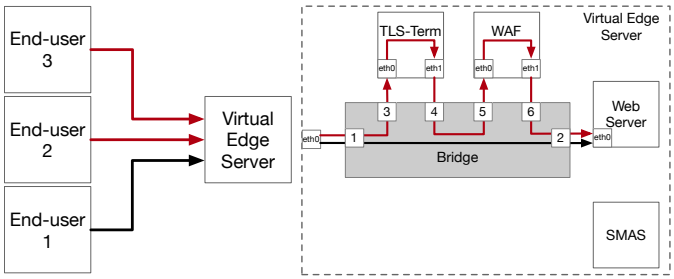
Details. This demonstration scenario begins with 3 end-users requesting for a video stream. End-user 1, the legitimate end-user, streams the HLS formatted video by acquiring the manifest file and issuing the corresponding chunk requests. Once the manifest file is received, End-user 2, an abusive end-user, starts issuing frequent requests for the same video chunk during short time intervals. End-user 3, another abusive end-user, requests a group of video chunks repeatedly. An abnormal rate of manifest file or chunk requests from one or several end-users raises a *suspicious_ip* alert. Such an alert triggers Policy 4 in the orchestrator. As shown in Figure 4b, the orchestrator deploys a chain composed of two security functions: (1) a TLS termination which terminates TLS sessions, and (2) a ModSecurity Web Application Firewall (WAF) [1] programmed with rate-limiting policies in the application layer. These rules instruct the WAF to limit VoD requests per-IP. According to these rules a Web Server cannot deliver (to an identical IP):

- a video chunk more than 2 times per 10 seconds
- a group of 4 video chunks more than once per 5 seconds

The parameters (i.e., the permitted rate of identical requests per-IP) in the rate-limiting rules can be configured on a per-media basis. For instance, chunks with a higher bit-rate could be allowed to be requested less frequently (e.g., n chunks downloadable every $n \times d$ time interval, where d is the average length of chunks calculated from the manifest file), whereas chunks with lower bitrates could be allowed to be downloaded more frequently (n chunks every 0.5 $n \times d$).

IV. Conclusion

This paper described the demonstration of a configurable security system developed to protect edge-servers. This system behavior is governed by high-level policies the enforcement of which results in the deployment of security function chains. This deployment is achieved dynamically and automatically. We illustrated the system architecture and demonstrated how our system can be flexibly programmed to mitigate two real-world threats. In the first demonstration, our system mitigates a network layer flooding attack by deploying a chain consisting of a rate-limiting function. We show how our system immediately recovered the degraded throughput of legitimate traffic. In the second demonstration scenario, an application layer abusive behavior is immediately rate-limited by deploying a security chain including a TLS termination and a WAF that is configured to rate-limit or block abusive requests.



(a) Demo Setup (b) Mitigation

Fig. 4: Application Layer Rate Limiting

$$\begin{aligned} &suspicious\_ip \textbf{ initiates } create\_chain(l: \\ &\quad \text{“not src net 99.231.0.0/16”}, 1, 2, \\ &\quad \{t : TLS\text{-}Term, w : WAF\}) \\ &\quad \textbf{if not } chain(l) \end{aligned} \tag{4}$$

Fig. 5: Application Layer Rate Limiting Policies

References

[1] Modsecurity, an open source, cross-platform web application firewall (waf). https://www.modsecurity.org.

[2] Open vswitch. http://openvswitch.org.

[3] Cloudflare rate limiting. https://goo.gl/PovNvK, 2017.

[4] What is rate limiting? https://goo.gl/HxWRC9, 2017.

[5] T. Alharbi, A. Aljuhani, and H. Liu. Holistic ddos mitigation using nfv. In *2017 IEEE CCWC*, 2017.

[6] C. Baral, J. Lobo, and G. Trajcevski. *Formal characterizations of active databases: Part II*, pages 247–264. Springer Berlin Heidelberg, 1997.

[7] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey. Bohatei: Flexible and elastic ddos defense. In *USENIX Conference on Security Symposium*. USENIX Association, 2015.

[8] A. H. M. Jakaria, W. Yang, B. Rashidi, C. Fung, and M. A. Rahman. Vfence: A defense against distributed denial of service attacks using network function virtualization. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, 2016.

[9] E. Jalalpour, M. Ghaznavi, D. Migault, S. Preda, M. Pourzandi, and R. Boutaba. A security orchestration system for cdn edge servers. In *2018 IEEE Conference on Network Softwarization (NetSoft)*, June 2018.

[10] J. Li, S. Berg, M. Zhang, P. Reiher, and T. Wei. Drawbridge: Software-defined ddos-resistant traffic engineering. In *ACM*. ACM, 2014.

[11] D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, page 2, 2014.

[12] P. Quinn, U. Elzur, and C. Pignataro. Network Service Header (NSH). Internet-Draft draft-ietf-sfc-nsh-28, IETF, 2017. Work in Progress.

[13] M. Yu, Y. Zhang, J. Mirkovic, and A. Alwabel. Senss: Software defined security service. In *ONS*, Santa Clara, CA, 2014. USENIX.