Automated Distribution of Access Control Rules in Defense Layers of an Enterprise Network

Adam Pavlidis, Marinos Dimolianis, Dimitris Kalogeras, Vasilis Maglaris
Network Management & Optimal Design Laboratory (NETMODE)
School of Electrical & Computer Engineering,
National Technical University of Athens (NTUA), Greece
{apavlidis, mdimolianis, dkalo, maglaris} [at] netmode.ntua.gr

***Abstract*— In this demo paper we present a network management framework for the automated mitigation of multi-vector anomalies. Our approach leverages on Salt to define and distribute system-specific Access Control Rules to network devices and hosts, in a streamlined device-agnostic manner. Network devices are managed using NAPALM, a library offering high-level programmable interfaces via different southbound protocols, e.g. NETCONF, SSH, HTTP. Our Proof-of-Concept testbed incorporates two hardware devices, and two end hosts used accordingly as the attacker and the victim of a multi-vector DDoS attack. As part of the demo, we will generate a DDoS attack and showcase the capabilities offered by the proposed platform towards the attack mitigation.**

Keywords*— *DDoS Mitigation, Access Control Rules, Salt, Jinja2, NAPALM

I. INTRODUCTION

Distributed Denial of Service (DDoS) attacks are ever growing in terms of scale and sophistication. This is highlighted in recent attacks [1] that use multiple vectors in an attempt to overwhelm their target and severely disrupt Internet services. Such attacks are either redirected in specialized high-performance cloud infrastructures [2] or require dedicated on-premise hardware/software appliances. Modern approaches tend to build on the Software-Defined Networking (SDN) and/or Network Function Virtualization (NFV) paradigms to deploy and scale mitigation pipelines capable of sustaining severe network attacks reaching rates of Tbps.

In [3], we proposed an access control schema that mitigates multi-vector DDoS attacks by distributing mitigation tasks across an enterprise network. Specifically, our mechanism utilized the firewall capabilities of various devices, located in different hierarchical network layers (stages) to deploy Access Control Rules (ACRs). This was formulated as a Generalized Assignment Problem (GAP). Assigned ACRs were distributed using various southbound protocols supported by the *Ryu* SDN Controller [4]. In this demo, we illustrate modern network automation techniques via which we orchestrate and enforce access control policies.

II. DISTRIBUTION OF ACCESS CONTROL RULES: A NETWORK AUTOMATION APPROACH

Our approach is based on the *Salt* automation platform [5] to distribute ACRs in a streamlined, device-agnostic manner. The key elements in the *Salt* architecture are the *Salt master* and the *Salt minions*. *Minions* may be perceived as software agents deployed within various Operating Systems; communication with the *master* is achieved over a publish-subscribe messaging system. At the master's behest, a minion executes Python modules (execution modules) or enforces a state (e.g. firewall configuration) defined in *SaLt State* (*SLS*) files. In addition, *Jinja2* templates [6] are used to render *minion*-specific configuration and/or *SLS* files. These along with *minion*-specific (*pillar*) data, are distributed from the *master* via the *Salt* messaging system. Note that, most operations are published to the *minions* and executed in a distributed fashion.

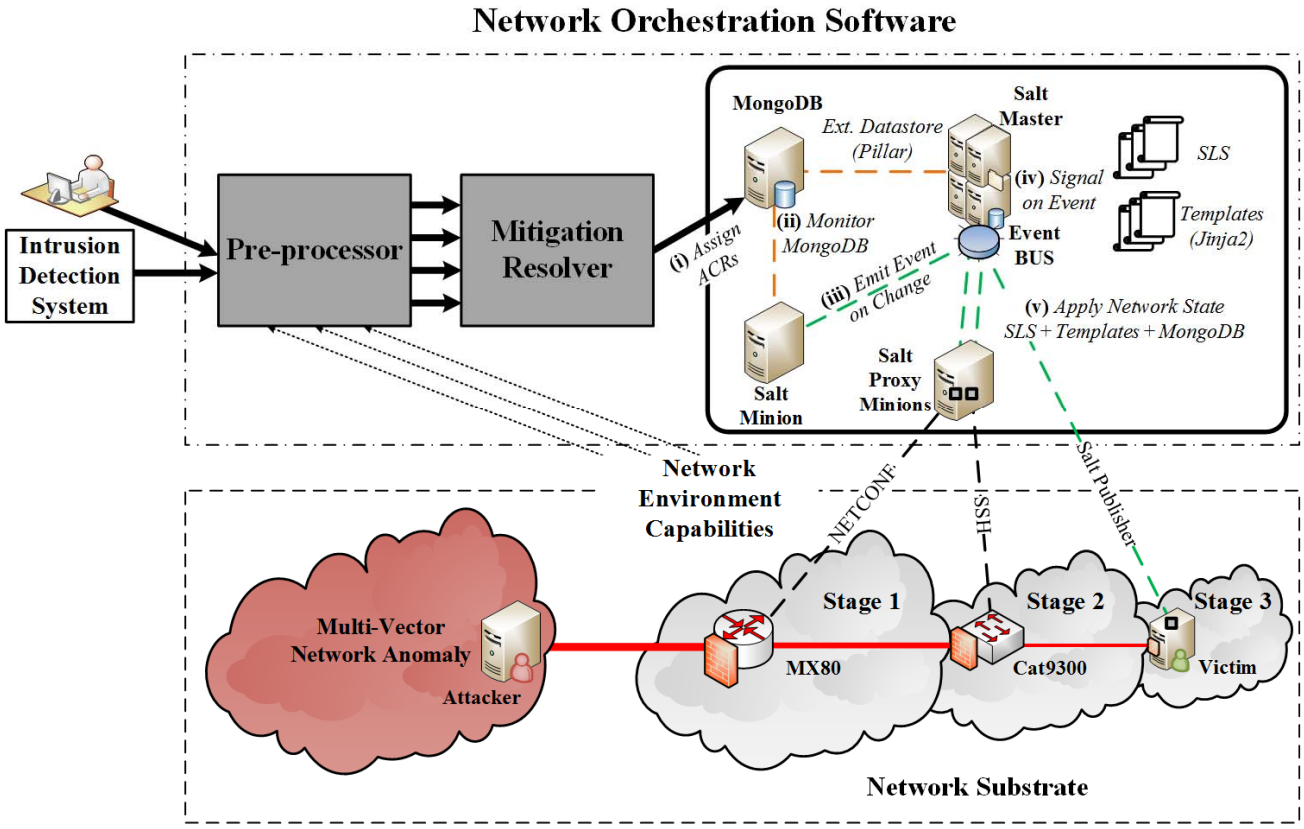
Notably, network devices are usually limited by vendor restrictions and cannot host a *Salt minion*; thus support is offered via specialized *Proxy minions* deployed in general purpose systems (e.g. Linux). These operate as regular *Salt minions* and interface with the devices via a specific *driver*. To that end we utilize *NAPALM* [7], a Python library providing high-layer abstractions for device/vendor agnostic programmability. Specifically, we load configuration commands into network devices, rendered from *Jinja2* device-specific templates.

Depicted in the figure below, is our proof-of-concept testbed comprised of a typical network substrate, controlled / managed by our network orchestration software.

The network substrate consists of two hosts interconnected by two hardware network devices; one host transmits (attacker) and the other receives (victim) a multi-vector DDoS attack. The victim and the two network devices provide an abstraction for a typical enterprise network. The attacker is connected to a Juniper MX80-48T router. In turn, this device is directly connected with a Cisco IOS Catalyst 9300 that operates as a L2/L3 switch. Both the attacker and the victim are instantiated as Linux Servers. The MX80, Cat9300 and the victim represent Defense Stages 1, 2 and 3 respectively. During a detected DDoS attack, the mitigation process deploys: (a) *firewall terms* for the MX80, (b) *access control lists* for the Cat9300 and (c) *iptables rules* for the Linux server operating as the victim.

This work was partially supported by the European Commission Horizon 2020 Framework Programme for Research and Innovation, Grant Agreement No. 731122 (GN4-2, GÉANT Project).

978-3-903176-15-7



Demo Testbed – DDoS Mitigation using Salt and NAPALM

Our network orchestration software via the Pre-processor correlates (a) the detected malicious sources for an attack vector, (b) the operator's preferences and (c) network environment capabilities to formulate the Generalized Assignment Problem (details are available in [3]). Subsequently, the mitigation resolver computes a feasible solution that meets device constraints and operator preferences. Based on this solution, it assigns ACRs to devices in different Defense Stages.

As shown in the figure, ACRs are inserted in a MongoDB, as distinct *documents*. A *document* associates malicious sources with the device (network or host) on which ACRs should be deployed. Furthermore, the MongoDB is also used by *Salt* as an external *Pillar*, i.e. a data store that maintains *minion* specific data. Note that a device is directly mapped to a distinct *minion*. An additional *minion* continuously monitors the MongoDB and upon change triggers an appropriate event. This event is published in the *Salt* event bus and received by the *master*. In turn, the *master* instructs *minions* to apply the desired state (i.e. *ACRs* for each device) based on: (a) *Pillar* data i.e. malicious IPs, (b) *Jinja2* templates that describe *access control list*, *firewall* terms, or *iptables* rules and (c) *SLS* files defining the configuration rendering and distribution process.

Proxy minions hosted within a dedicated server convey mitigation policies to network substrate devices via southbound protocols, i.e. NETCONF, SSH. We utilize *NAPALM* as the *proxy minion* driver that employs the *eznc* library (NETCONF-based) for the MX device and the *netmiko* library (SSH-based) for the Cat9300 accordingly.

III. Demonstration

Our demonstration illustrates the automated distribution of ACRs in Defense Layers of an Enterprise Network; our goal is to efficiently mitigate a multi-vector DDoS attack from multiple malicious sources. Note that we do not consider operational characteristics of the Intrusion Detection System (delay, accuracy) which are beyond the scope of this demo. The demo will showcase the mitigation workflow step-by-step and highlight key component interactions. This includes displaying: (a) malicious IP updates in MongoDB, (b) events and instructions formatted in JSON, (c) SLS file structure, (d) device and host configuration templates. Furthermore, we will graphically display traffic metrics of the attack traffic during the evolution and mitigation of the attack.

References

[1] "The rise of multivector DDoS attacks", available at: https://blog.cloudflare.com/the-rise-of-multivector-amplifications/, last visited November 2018

[2] "Akamai Prolexic Solutions", avaialable at: https://www.akamai.com/us/en/products/cloud-security/prolexic-solutions.jsp, last visited November 2018

[3] M. Dimolianis, A. Pavlidis, D. Kalogeras and V. Maglaris, "Mitigation of Multi-vector Network Attacks via Orchestration of Distributed Rule Placement", to appear in Proceedings of IM2019, Washington D.C., USA, April 2019

[4] "Ryu the Network Operating System (NOS)", available at: https://ryu.readthedocs.io/en/latest, last visited November 2018

[5] "Saltstack Platform", available at: https://saltstack.com, last visited November 2018

[6] "Jinja2 Template Language", available at: http://jinja.pocoo.org/docs/2.10, last visited November 2018

[7] "NAPALM library", available at https://napalm-automation.net, last visited November 2018