A demonstration of an autonomous approach for cyberattack mitigation

Original
A demonstration of an autonomous approach for cyberattack mitigation / Pizzato, Francesco; Bringhenti, Daniele; Sisto, Riccardo; Valenza, Fulvio. - ELETTRONICO. - (In corso di stampa). (Intervento presentato al convegno 2025 21st International Conference on Network and Service Management (CNSM) tenutosi a Bologna (IT) nel 27-31 October 2025).

Availability:
This version is available at: 11583/3003223 since: 2025-09-21T06:34:43Z

Publisher:
IEEE/IFIP

Published
DOI:

A demonstration of an autonomous approach for cyberattack mitigation

Francesco Pizzato, Daniele Bringhenti, Riccardo Sisto, Fulvio Valenza
Dip. Automatica e Informatica, Politecnico di Torino, Torino, Italy, Emails: {first.last}@polito.it

***Abstract*—The increasing complexity and size of virtual networks, jointly with the fast-evolving nature of modern threats, have significantly amplified the challenge of mitigating cyberattacks in real time. In particular, these factors have made the traditional approaches for network security reconfiguration unfeasible, as they rely heavily on manual operations. To address these issues, this demo presents a looping process that autonomously mitigates ongoing attacks by extracting security policies from intrusion detection system alerts and automatically reconfiguring distributed firewalls via a provably correct and optimized approach. The proposed system architecture is composed of several interconnected components responsible for the full lifecycle from the detection of an attack to the deployment of the updated and secure configuration, operating in a fully automated and self-triggering way, aiming to reduce human involvement while improving mitigation speed and correctness.**

***Index Terms*—network security, security automation, attack mitigation**

I. INTRODUCTION

Recent years have witnessed significant innovations in networked systems, driven by emerging paradigms such as network softwarization and cloud computing. As a result, modern networks show levels of dynamism and flexibility that were previously unattainable. This evolution, combined with the growing need for globally distributed applications, has led to a substantial increase in network size and complexity. In parallel, adversaries have adapted and refined their techniques to exploit these characteristics for their gain [1].

To face such cyberattacks, one of the most effective countermeasures remains the reconfiguration of distributed firewalls, which continue to serve as the first line of defense. However, the traditional approaches for their reconfiguration rely on human-based trial-and-error strategies, which are slow and prone to misconfiguration errors. To address this problem, several solutions have been proposed in literature to automate the firewall reconfiguration process. A common strategy involves policy-based management, allowing administrators to express desired connectivity requirements, i.e., the sets of denied and allowed traffic, through high-level user-friendly policies. These statements are then refined by automated tools into the low-level configuration tailored to the specific firewall technology.

Despite these advancements, the state-of-the-art solutions still depend significantly on human involvement [2]. For example, in many approaches, administrators are required to manually interpret alerts raised by monitoring agents, translate them into appropriate policies, and input them into automated tools. This multi-step human-centric workflow introduces delays in attack mitigation, limiting the effectiveness of automated response mechanisms and increasing the risk of system compromise. Moreover, such reliance on human-centered processes increases the likelihood of misconfigurations, particularly in crisis-driven scenarios of impending cyberattacks.

This paper demonstrates an approach, originally presented in [3], [4], for an autonomous, optimized, and provably correct process for reconfiguring a distributed firewall system in the event of an attack detected by intrusion detection system (IDS) solutions. The goal is to reduce human intervention as much as possible by fully automating the cyberattack mitigation workflow, thereby ensuring a faster and more reliable response.

The remainder of the paper is structured as follows. Section II presents the architecture, describing all the involved modules of the automated looping approach for autonomous cyberattack mitigation. Then, Section III presents the demonstrations and details what will be showcased. Finally, Section IV concludes the paper and discusses future research directions.

II. SYSTEM ARCHITECTURE

The proposed solution implements a fully autonomous, self-triggering loop for cyberattack mitigation through the reconfiguration of distributed firewalls. The workflow, illustrated in Fig. 1, is designed as the concatenation of several modules that implement the different phases of cyberattack mitigation. These are designed to operate continuously without human intervention once deployed, seamlessly integrating detection, reaction, and enforcement functionalities.

1) *Input preparation*. This is the only phase requiring human intervention. The network administrator provides two inputs essential to trigger the autonomous monitoring and mitigation process: i) a set of Network Security Policies (NSPs), defining traffic flows that must either be blocked due to potential threats or allowed to preserve service connectivity; ii) a description of the network topology and current security configuration, including the deployment and rule sets of existing firewalls (satisfying the NSPs). Both are formally modeled in [4], while the corresponding details are omitted here for brevity.
2) *Firewall Configuration eXchange (FCX).* The FCX module converts the NSPs and the network description into low-level implementation-specific firewall rules (e.g., eBPF, iptables, Open vSwitch). These configurations are

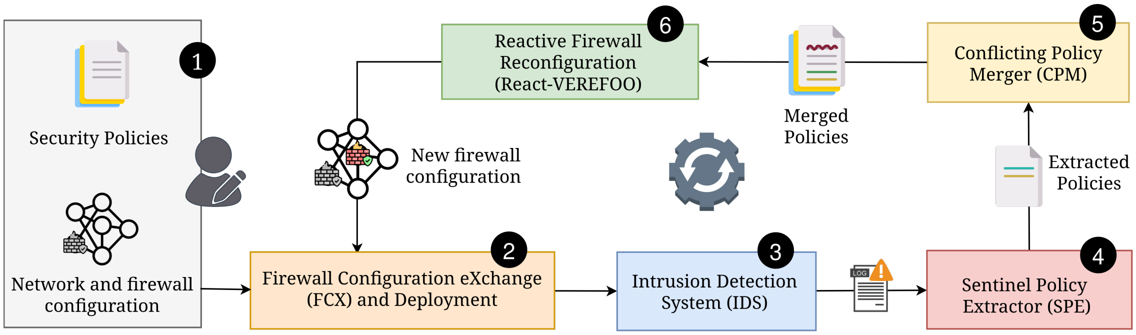
1
Security Policies
Network and firewall configuration
New firewall configuration
6
Reactive Firewall Reconfiguration (React-VEREFOO)
Merged Policies
5
Conflicting Policy Merger (CPM)
Extracted Policies
2
Firewall Configuration eXchange (FCX) and Deployment
3
Intrusion Detection System (IDS)
LOG
4
Sentinel Policy Extractor (SPE)

Fig. 1: Workflow for autonomous cyberattack mitigation.

then handed off to the network orchestrator for deployment within Virtual Network Functions (VNFs).

3) *Intrusion Detection System (IDS).* Once the initial security configuration is deployed, the network satisfies the NSPs. IDS agents, deployed as VNFs in the network, continuously monitor the traffic to detect potential attack patterns. When suspicious activity is identified, alerts are generated and recorded in IDS log files, containing information to identify the suspicious traffic flows.
4) *Sentinel Policy Extractor (SPE).* The component periodically checks the log files produced by IDSs for new alerts. When a valid alert is detected, the module applies some designed policy extraction algorithms to generate new NSPs aimed at blocking the malicious traffic associated with the detected attacks.
5) *Conflicting Policy Merger (CPM).* Newly extracted NSPs are merged with the current active policy set. A conflict resolution mechanism ensures consistency, resolving overlapping or contradictory rules (e.g., a new deny policy overlapping with an existing allow policy).
6) *Reactive Firewall Reconfiguration.* The updated set of NSPs is processed by a reconfiguration engine that computes the optimal firewall allocation and rule set needed to satisfy all current constraints, i.e., the merged set of NSPs. The engine leverages React-VEREFOO [3], an existing approach supporting correctness, optimization, and formal verification of firewall configuration.

Finally, the process operates in a continuous loop: IDSs continuously monitor the network, and new alerts trigger the re-evaluation and update of firewall rules, ensuring prompt, autonomous threat mitigation. Moreover, the proposed architecture is modular and extensible, allowing integration with different IDSs, orchestrators, and firewall backends.

III. DEMO

The demo showcases the implementation of the presented approach and its behavior in the event of a simulated ICMP flood-based Denial-of-Service (DoS) attack. In this demo, network models and NSPs are defined with the Medium Security Policy Language (MSPL), an XML-based format validated in various EU projects [5]. The FCX component, developed in Java with the JAXB library, maps MSPL files to backend-specific configurations for iptables, Open vSwitch, or eBPF firewalls. The SPE and CPM modules, in Python, exchange data via dedicated interfaces. React-VEREFOO, also in Java, provides a RESTful web interface. The entire system runs on a Docker Compose-based platform for network virtualization. The demo is designed as a sequence of steps, letting the audience experience all the modules presented in Section II.

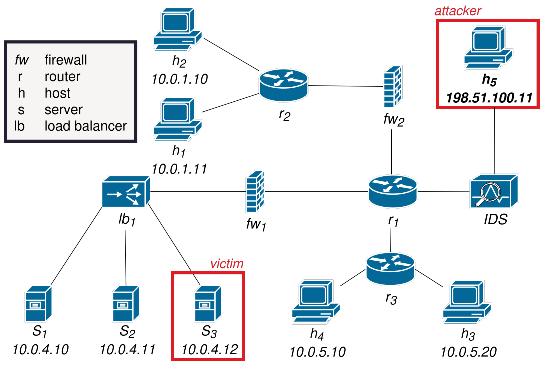

Fig. 2: Network topology and Firewall configuration.

The first step of the demo consists of the definition and deployment of the initial set of NSPs and the related network topology, with its firewall configuration. On the one hand, the NSPs and network topology are defined by the user in such a way that each NSP specifies a traffic, identified with the IP 5-tuple fields, and an action that should be applied to that traffic, i.e., allow or deny. On the other hand, in this demo, the firewall configuration is computed starting from the network topology and the NSPs by leveraging the automated firewall configuration tool itself, in its default version, VEREFOO, not focused on reconfiguration [3], but intended for a clean, initial network configuration. Instead, for reconfiguration, e.g., adding or removing a few NSPs to mitigate an attack, the React-VEREFOO version will be preferred for its enhanced performance and ad-hoc optimizations. The resulting network, represented in Fig. 2, consists of three subnets hosting private services whose access is monitored through an IDS. Within the demo, the attacker represents an internal host, either malicious or compromised, that wants to interfere with the operations of one of the private services, causing a DoS attack.

Second, the network and firewall configurations are passed as XML files to the FCX module through REST APIs. FCX handles the parsing and creation of a series of files, deployed through Docker Compose. As an example, a fragment of the XML file for the FW_1 configuration is provided in Listing 1.

```
<node id="4" name="fw_1" functional_type="FIREWALL">
  <neighbour id="107" name="r_1" />
  <neighbour id="108" name="lb_1" />
  <configuration name="AutoConf">
    <firewall defaultAction="ALLOW">
      <elements>
        <action>DENY</action>
        <source>10.0.4.11</source>
        <destination>10.0.5.10</destination>
        <protocol>TCP</protocol>
          <src_port>0-65535</src_port>
          <dst_port>0-65535</dst_port>
      </elements>
      <elements>
        <action>DENY</action>
        <source>10.0.4.11</source>
        <destination>198.51.100.11</destination>
        <protocol>OTHER</protocol>
          <src_port>0-65535</src_port>
          <dst_port>0-65535</dst_port>
      </elements>
              ...
    </firewall>
  </configuration>
</node>
```

Listing 1: Generated Firewall Configuration.

The FCX module handles the translation of XML configuration files to iptables rules. For instance, the configuration of FW_1 described by Listing 1 produces the script in Listing 2.

```
#!/bin/sh
cmd="sudo iptables"
${cmd} -F
${cmd} -P INPUT ACCEPT
${cmd} -P FORWARD ACCEPT
${cmd} -P OUTPUT ACCEPT
${cmd} -A FORWARD -p tcp -s 10.0.4.11/32 -d 10.0.5.10/32 --
     sport 0:65535 --dport 0:65535 -j DROP
${cmd} -A FORWARD -p icmp -s 10.0.4.11/32 -d
     198.51.100.11/32 -j DROP
...
```

Listing 2: Translated iptables configuration

Third, the monitoring agent is properly initialized by configuring a custom rule, reported in Listing 3, for *IDS*, a VNF running Snort. This produces an alert if more than 10 ICMP packets are received in one second. The user could trigger it by simulating the DoS attack by issuing the command `ping -i 0.05 10.0.4.12` within the attacker container.

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP
     flood attempt"; detection_filter:track by_src, count
     10, seconds 1; sid:1000001; rev:1;
```

Listing 3: Custom IDS rule for SNORT.

Fourth, once the attack is detected, an alert is raised from the IDS log file and is sent to the SPE module. This module processes the alert and uses a custom algorithm to extract a new NSP to block the traffic corresponding to the detected DoS attack, reported in Listing 4.

```
<PropertyDefinition>
   <Property graph="0" name="IsolationProperty" src="
        198.51.100.11" dst="10.0.4.12" lv4proto="OTHER"/>
</PropertyDefinition>
```

Listing 4: Extracted Network Security Policy.

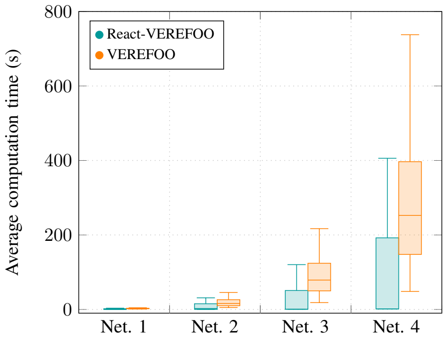

Fig. 3: React-VEREFOO scalability

Fifth, the CPM module handles the update of the current NSP set by introducing the extracted one in an anomaly-free way. The merged NSP set is passed to React-VEREFOO, along with the current network and firewall configuration. React-VEREFOO computes, within a short time, an updated firewall configuration to satisfy the updated MSP set. In the demo, only the configuration for FW_1 is recomputed, while other configuration elements, e.g., FW_2, are untouched. Fig. 3 also shows how React-VEREFOO improves computation time by a factor of 60% when compared to the non-optimized VEREFOO version for networks of increasing size, i.e., from 200 NSPs and 40 endpoints up to 500 NSPs and 100 endpoints, in a scenario where 30% of the total NSPs are modified.

Finally, the new firewall configuration is passed to the FCX module, so that the overall process can loop and apply the produced countermeasure to stop the attack.

IV. CONCLUSION AND FUTURE WORKS

This paper demonstrates a complete looping cyberattack mitigation process based on optimized and automated firewall reconfiguration. The modular approach minimizes human intervention by automating detection, analysis, and response. Future work would extend it to other firewall types, e.g., web application firewalls, and topology changes, e.g., link failures.

ACKNOWLEDGMENT

This work was supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the EU - NextGenerationEU.

REFERENCES

[1] Proton, “A brief update regarding ongoing DDoS incidents,” Available: https://proton.me/blog/a-brief-update-regarding-ongoing-ddos-incidents, (Visited: 2025-07-28), 2022.

[2] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, “Automation for network security configuration: State of the art and research trends,” *ACM Comput. Surv.*, vol. 56, no. 3, pp. 57:1–57:37, 2024.

[3] F. Pizzato, D. Bringhenti, R. Sisto, and F. Valenza, “Automatic and optimized firewall reconfiguration,” in *NOMS 2024 IEEE Network Operations and Management Symposium, Seoul, Republic of Korea, May 6-10, 2024.* IEEE, 2024, pp. 1–9.

[4] D. Bringhenti, F. Pizzato, R. Sisto, and F. Valenza, “Autonomous attack mitigation through firewall reconfiguration,” *International Journal of Network Management*, vol. 35, no. 1, p. e2307, 2025.

[5] J. M. B. Murcia, A. M. Zarca, and A. F. Skarmeta, “BASTION: beyond automated service and security orchestration for next-generation networks,” *Comput. Networks*, vol. 267, p. 111352, 2025.