Automating Mitigation of Amplification Attacks in NFV Services

Matteo Repetto, Gianmarco Bruno, Jalolliddin Yusupov, Guerino Lamanna, Benjamin Ertl, and Alessandro Carrega, *Member, IEEE*

***Abstract*—The combination of virtualization techniques with capillary computing and storage resources allows the instantiation of Virtual Network Functions throughout the network infrastructure, which brings more agility in the development and operation of network services. Beside forwarding and routing, this can be also used for additional functions, e.g., for security purposes. In this paper, we present a framework to systematically create security analytics for virtualized network services, specifically targeting the detection of cyber-attacks. Our framework largely automates the deployment of security sidecars into existing service templates and their interconnection to an external analytics platform. Notably, it leverages code augmentation techniques to dynamically inject and remove inspection probes without affecting service operation. We describe the implementation of a use case for the detection of DNS amplification attacks in virtualized 5G networks, and provide extensive evaluation of our innovative inspection and detection mechanisms. Our results demonstrate better efficiency with respect to existing network monitoring tools in terms of CPU usage, as well as good accuracy in detecting attacks even with variable traffic patterns.**

***Index Terms*—Amplification attacks, DDoS, ARIMA, eBPF, NFV, 5G.**

I. Introduction

NETWORK Function Virtualization (NFV) has represented a ground-breaking innovation in the rather static domain of Telco operation, bringing unprecedented opportunities for agile development and provisioning of virtualized Network Services (NSs). Beside the mere virtualization of network functions, automation has been a key driver since the beginning, pursuing policy-driven management and control of NSs.

Notably, the scope for NFV has not been restricted to packet forwarding and routing, but has also looked for new forms of functions that could be more effectively implemented in the network than in end devices [1], also including detection and mitigation of cyberattacks [2]. Indeed, several kinds of attacks, especially those concerning Denial of Service (DoS) can be mitigated in a more effective and efficient way by Telco providers on behalf of their customers. Even if detection and reaction capability could be easily integrated in the design of any NS, we argue that this approach would provide a quite static and rigid solution, and lack the necessary agility to adapt to the continuously evolving network threat landscape [3]. Therefore, we designed the ASTRID[1] framework for creating tailored detection and mitigation processes for virtualized NSs. It allows to augment descriptive service templates with security agents, to connect them to analytics platforms for analysis and correlation, and to define reaction and mitigation policies that automate response to cyberattacks.

In this paper, we describe the architecture of the ASTRID framework and demonstrate our solution for a specific use case, namely mitigation of Distributed DoS (DDoS) attacks. Specifically, our investigation focuses on amplification attacks originated by compromised botnets attached to 5G networks and the mechanisms to detect and stop them at the edge. The main innovations of our work with respect to existing State of the Art include:

- the use of security sidecars for monitoring and inspection, which preserve the integrity of software images of Virtual Network Functions (VNFs);
- the decoupling of security processes from service management by a dedicated security Dashboard, which allows better separation of concerns between different teams (devops and security staff);
- the development of inspection and enforcement programs in the eBPF framework, including a generic control plane (*Dynmon*) that allows to load and remove them at runtime without affecting service continuity;
- the Analytics ToolKit (ATk), which is a detection tool that combines coarse and fine grained measurements to properly balance the effectiveness of Deep Packet Inspection (DPI) with the efficiency of simple protocol counters.

Manuscript received 16 November 2021; revised 8 March 2022; accepted 28 April 2022. Date of publication 5 May 2022; date of current version 12 October 2022. This work has received funding from the European Commission, Grant Numbers: 786922 (ASTRID) and 833456 (GUARD). The associate editor coordinating the review of this article and approving it for publication was C. Hesselman. *(Corresponding author: Matteo Repetto.)*

Matteo Repetto is with the IMATI Institute, National Research Council of Italy (CNR), 16149 Genoa, Italy (e-mail: matteo.repetto@ge.imati.cnr.it).

Gianmarco Bruno is with the Product Development Unit OSS, Ericsson Telecomunicazioni, 16151 Genoa, Italy (e-mail: gianmarco.bruno@ericsson.com).

Jalolliddin Yusupov is with the Department of Automatic Control and Computer Engineering, Turin Polytechnic University, Tashkent 100095, Uzbekistan (e-mail: jaloliddin.yusupov@polito.uz).

Guerino Lamanna is with the Research and Development Department, Infocom Srl, 16121 Genoa, Italy (e-mail: guerino.lamanna@infocomgenova.it).

Benjamin Ertl is with Agentscape AG, 14199 Berlin, Germany (e-mail: b.ertl@agentscape.de).

Alessandro Carrega is with SSN Lab, Italian National Inter-University Consortium for Telecommunications (CNIT), 16145 Genoa, Italy (e-mail: alessandro.carrega@cnit.it).

Digital Object Identifier 10.1109/TNSM.2022.3172880

[1]AddreSsing Threats for viRtualIseD services, https://www.astrid-project.eu/.

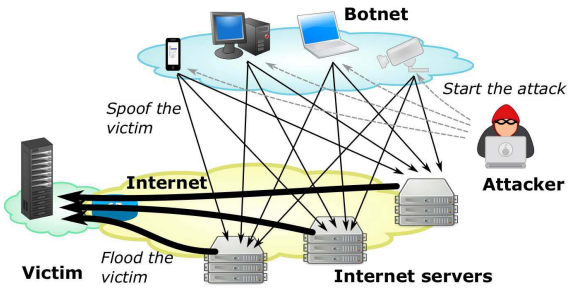

Fig. 1. Typical attack pattern for volumetric DDoS with amplification.

Remarkably, our approach addresses the most recent challenges in NFV, which is currently moving towards Platform-as-a-Service environments. The possibility to run different eBPF programs with a common agent goes in the direction of truly agentless systems, which could be further extended to serverless environments as well.

In a preliminary paper [4], we already described monitoring and detection of amplification attacks at the network edge. Here, we extend our previous work by i) automating the deployment and configuration of security agents; ii) extending the scope to mitigation of the attack; iii) performing extensive performance analysis on effectiveness and efficiency aspects, which encompasses the quality of the detection, overhead of security agents, and response latency. Our results show that the ATk can distinguish between periodic fluctuations of requests and anomalies. The overhead for collecting the measurements is also quite limited, and the overall framework does not introduce relevant delays in the detection process.

The rest of this paper is organized as follows. We discuss the problem of amplification attacks in Section II, which includes existing mitigation techniques and prospective usage of NFV for this purpose. We then describe the architecture, scope, and workflows of the ASTRID framework in Section III. The implementation of the use case is described in Section IV, including the details of the deployed agents, the analytics engine, and the response policies. We report numerical results from our performance analysis in Section V, which includes comparison with alternative monitoring tools. We revise related work in Section VI. Finally, we give our conclusion in Section VII.

II. Amplification Attacks

DDoS attacks still represent one of the most challenging security threat for global organizations; indeed, the vast increases in data traffic expected from 5G and the growing number of (vulnerable) IoT devices give attackers better opportunity for launching larger attacks [5], [6]. Volumetric DDoS are able to saturate the Internet pipe of even the largest organizations (e.g., GitHub).[2] To reach the necessary amount of traffic, they combine two complementary techniques, namely *distributed generation* of traffic from a huge number of compromised devices and *amplification* of the volume by hundreds or thousands of buggy or misconfigured servers on the Internet.

The typical attack pattern is shown in Fig. 1. An attacker usually uses a botnet of compromised nodes. These nodes query a large number of servers, while spoofing the IP address of the victim. As a result, such servers flood the victim with responses that are much larger in size than the original requests, hence generating the amplification effect. Well-known examples of vulnerable protocols include the Network Time Protocol (NTP) and the Domain Name System (DNS). Beside these examples, the problem extends to other servers and protocols as well (Memcache, SIP, LDAP, RIP, SNMP). The relevant parameter to size the impact of the attack is the "amplification factor," namely how many times the response is bigger than the original query; it may range from a few to thousand times [4].

A. Existing Mitigation Strategies

The detection of volumetric amplification attacks is trivial, because a single server or an entire network are flooded by an anomalous amount of traffic. However, mitigation is more challenging, because packets come from legitimate sources and carry valid data. There are currently two different approaches for this purpose: inline solutions and scrubbing centers.

Inline solutions work at the boundary of the victim's network. The detection is usually more accurate, because deep packet inspection is used to identify known attack patterns, whereas scrubbing centers usually consider aggregate statistics. However, they prevent a single server or the entire network to become flooded, but cannot avoid the access link to saturate. Moreover, large investment (usually in inflexible hardware) is requested by each organization, which remains underutilized for long periods of time.

Diversion of suspicious traffic towards scrubbing centers prevents access links to become saturated [7]. This solution limits the impact of the attack to the time needed to detect the saturation and to divert packets; the latter is typically done by changing BGP routing or DNS resolution and usually requires a few dozens of minutes at most. However, there are non-negligible effectiveness and efficiency issues for scrubbing centers. The capacity of the scrubbing center must be sized some times more than the biggest expected attack (e.g., four to ten), and it should be continuously increased as attackers increase the maximum volume of their attacks.[3] Additionally, scrubbing center solutions typically sample the traffic flows and make aggregate statistics, hence detecting general patterns that may lead to over-blocking legitimate traffic users. Definitely, even this approach is largely inefficient and partially ineffective, because the overall Telco's infrastructure is still overwhelmed by large volumes of malicious traffic, scrubbing centers must be largely overdimensioned, and the duration of the denial of service experienced by the victim is not negligible.

B. Mitigation in NFV

The most effective defensive mechanism against DDoS would be to detect and mitigate it at the origin. Stopping

[2]https://www.wired.com/story/github-ddos-memcached/

[3]Less than two years after the unprecedented 1.35 Tbps DDoS attack experienced by GitHub, AWS reported to have defended against a 2.3 Tbps attack in February 2020, which almost doubled the previous volume. During the same period, Imperva reported one of its client to have experienced a 500 million packets-per-second attack, which represents the most intensive DDoS attack against network infrastructure in the history of the Internet.

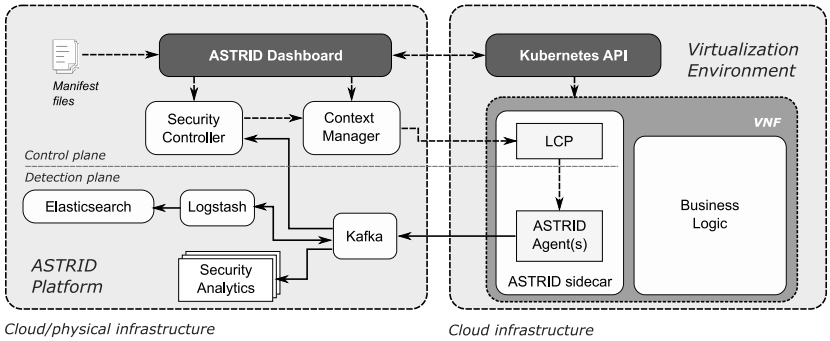

Fig. 2. The ASTRID framework.

amplification attacks before they are amplified would be possible, in theory, by applying safe configurations to servers, blocking unnecessary ports, activating anti-spoofing filters; however, this is difficult to achieve in practice, because it depends on each organization that connects servers and devices to the Internet. An even more efficient yet challenging approach would be to block DDoS as close as possible to its root, namely at the boundary of Telco's infrastructures. Unfortunately, the implementation of detection and mitigation mechanisms by Telco providers on behalf of their customers has been historically hindered by the lack of computing resources at the network edge and the prohibitive cost to deploy security appliances in a capillary way.

With the advent of NFV these obstacles can be easily overcome. The availability of computing and storage resources throughout the network allows the implementation of flexible monitoring, inspection, detection and enforcement tasks that were not possible with legacy hardware appliances. NFV can therefore provide scalable DDoS solutions close to the network edge, which are able to stop attacks close to their root. This makes inline DDoS detection and mitigation faster, more accurate, less expensive and keeps harmful traffic from getting past the edge into the core of the Telco network.

III. The ASTRID Framework

Even if detection and reaction capability could be easily integrated in the design of any NS, we argue that this approach would provide a quite static and rigid solution, and lack the necessary agility to adapt to the continuously evolving network threat landscape [3]. Therefore, we designed the ASTRID framework to implement detection and analytics processes on virtualized NSs. The main purpose is to simplify the instrumentation of VNFs with the necessary set of security agents, and to automate as much as possible the collection of data and the implementation of response actions. The implementation explicitly targets containerized applications, orchestrated by Kubernetes. This addresses the growing interest in Platform-as-a-Service (PaaS) virtualization, which has proven to be more efficient and agile than Virtual Machines in Infrastructure-as-a-Service environments [8]. This also allows to deploy security agents as sidecar containers, and even to leverage code augmentation as we did in ASTRID (see Section IV).

Fig. 2 shows the overall architecture of the ASTRID framework. It is made of a centralized platform (on the left side) and a security sidecar (right side) co-located in the same pod as the main business logic of the VNFs. In this architecture, security analytics are implemented in a common centralized platform, whereas the sidecars only includes monitoring and inspection tasks. The reasons behind centralized detection include the following considerations:

- The possibility to correlate data and events from multiple sources, which improves the detection accuracy.
- The design of lightweight monitoring and inspection functions that do not bring excessive overhead to PaaS/serverless environments.
- The need for persistent storage in virtualized environments already requires to collect data in a common location.

Both the centralized platform and the sidecars are split into a *Control plane*, which is responsible for configuration and control, and a *Detection plane*, which implements concrete monitoring and detection tasks. Differently from existing solutions, ASTRID is not conceived as a standalone cybersecurity tool per se; rather, it can be viewed as a sort of *middleware* for implementing many complementary detection processes for virtualized services. The detection of amplification attacks is just an example, which leverages ASTRID capabilities to extend existing NSs.

The main capabilities implemented by ASTRID include:

- definition of security processes for an existing NS, undertaken through the Dashboard;
- automatic configuration and response in the Control plane, managed by the Context Manager (CM) and the Security Controller (SC);
- security analytics in the Detection plane that analyze and correlate data, and create notifications and alerts.

In the following, we describe these capabilities in detail with reference to the relevant components in the platform.

A. Definition of Detection Processes

The first capability of the ASTRID framework concerns the definition of *Security Analytics Pipelines* (SAPs) which define three main elements chained to implement a detection service:

- the selection of security agents to be co-located with VNFs;

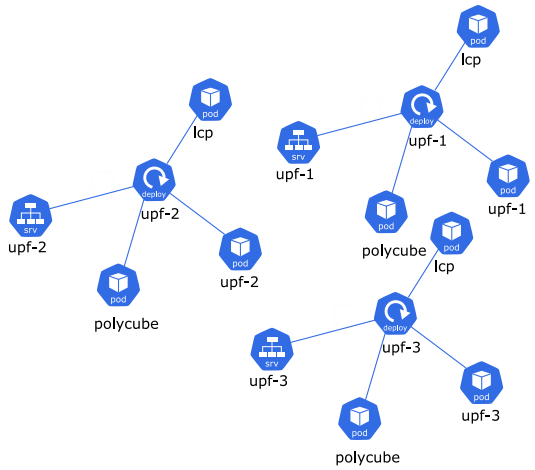

Fig. 3. Example of service topology visualized in the ASTRID Dashboard.

- the selection of analytics for data processing and correlation;
- the identification of response actions to be triggered by alerts sent by analytics.

The ASTRID Dashboard allows the definition of SAPs in a graphical way. This process starts by loading an existing NS, given by a Kubernetes manifest file. The Dashboard represents the service as a graph, where nodes corresponds to service pods and links to logical relationships between Kubernetes services, deployments, pods and containers (see Fig. 3). This graph is then enriched with security agents, which are linked to every pod that must be monitored. An analytics engine is then selected that will be fed with data collected from agents. The different types of security agents, as well as available analytics, are dynamically loaded from the Context Manager (see Section III-B). Finally, a set of response policies automate the behavior of the system in case of alarms and other relevant events. Response policies are user-defined set of rules that can be directly edited in the Dashboard or loaded from an existing file.

The outcome of the SAP definition process is an enriched manifest file, where security agents are embedded in a dedicated ASTRID sidecar. The sidecar is an additional container co-located in the same VNF pod as the function to be monitored. The modification of a Kubernetes manifest file is not trivial, because this often requires to share resources among containers (e.g., to read log files in the container hosting the main business logic). The current implementation mostly supports network probes for packet inspection. This is the most interesting features, because other data (e.g., logs) can already be collected by Kubernetes API and does not strictly require additional agents.

The ASTRID sidecar always contains a special kind of agent, named *Local Control Plane* (LCP), which acts as common control point to change the configuration of monitoring and enforcement agents. The LCP supports different configuration methods (YAML files, REST APIs, command line), which cover most of existing agents. As regard to security agents, a few "Beats" from the Elastic Stack have been integrated, although the main goal is to support code augmentation for instrumenting the Kernel (namely, the possibility to inject eBPF code, as described in Section IV). It is worth noting that log collection is already provided by Kubernetes; however, the availability of agents for this purpose fully decouples the framework from the underlying infrastructure, and makes it more portable to multi- and cross-cloud deployments.

Once the enriched template is available, the Dashboard can start service deployment, which is done by conventional Kubernetes API. After deployment, the Dashboard collects runtime parameters (e.g., IP addresses), which are necessary to configure security agents in the next step. The Dashboard therefore hands over configuration to the SC, by passing the SAP and runtime parameters. It is important to note that all software deployment and lifecycle management is performed by the Dashboard through Kubernetes API, whereas the SC only takes care of configuration matters. This provides separation of concerns between service management and security processes, preventing possible conflicts in software deployment between devops and security teams.

B. Automatic Configuration and Response

The smart core of the ASTRID platform is the Security Controller (SC), which automatically undertakes configuration and response actions. Configuration is triggered by the reception of a SAP descriptor from the Dashboard. It entails the configuration of security agents, security analytics, the other components of the data handling pipeline in the Detection plane (see Section III-C), and the SC itself (by loading the set of response policies).

The implementation of the SC is based on the Drools business rules management framework. Drools engine is in charge of simultaneously satisfying policies based on the Rete algorithm. For its functioning, Drools is based on the available data (which are called facts) and on the rules that have been defined; there is a rules engine which, upon the occurrence of certain conditions that correspond to imposed rules, launches the corresponding action. It is also possible to automatically generate the source file from decision tables that can be written, for example, in Microsoft Excel, thus further facilitating the work of non-technical personnel. This provides separation between the application logic and business rules: the rules on which the program works are completely separate from the code of the program itself, in this way the maintainability of the software is greater and without the risk of manipulating business rules.

The ASTRID Security Controller is developed using Spring Boot framework. The SC implements the following interfaces:

- REST APIs for interacting with ASTRID components;
- Kafka API to receive notifications.

The definition of response policies follows an Event-Condition-Action pattern, where one or more response actions are triggered when an event occurs, if some context conditions are satisfied. Events are generated by the security analytics, whereas conditions may include the current time, agent configuration, service topology. The scope of the actions includes both changes in the configuration of ASTRID agents as well as management requests to the external Kubernetes orchestrator.

The first type of response is useful for mitigation of DDoS attacks, and it is therefore implemented in our use case. The second type can be used in case compromised software must be replaced, or to find a different location for migration in case of attacks to the underlying infrastructure.

The Context Manager (CM) provides an abstraction of the virtual service, in order to decouple the ASTRID platform from the implementation of security agents. This abstraction includes the topology of the service, and suitable models for both the description of the execution environments (Virtual Machines, containers, lambda functions) and security agents that capture their capabilities (i.e., what data they provide and how) rather than their concrete software implementation. Internally, the CM translates the high-level agent model into concrete configuration actions, which are then applied locally through the LCP agent. This approach allows to change the implementation and interface of security agents without affecting the rest of the ASTRID platform. The CM exposes a uniform REST interface for changing the configuration of local agents and security analytics, independent of specific protocols and syntax.

C. Attack Detection

The whole detection process is implemented by a data handling pipeline which collects data from agents and delivers it to the ASTRID platform, where they are indexed and stored in a NoSQL database (Elasticsearch). In addition, data can also be directly delivered to one or more engines for security analytics, hence following a programming model that supports both streaming and off-line analysis.

The architecture of the ASTRID platform builds on and extends the well-known and proven Elastic Stack, by collecting events and measurements on a Kafka bus. The architecture also includes the possibility to run Logstash filters for data transformation, encoding, indexing, and so on.

Security analytics include any algorithm that processes, correlates, and analyzes data for the sake of detecting attacks. Their output is represented by informative events and alarms, which are shared with the Dashboard (for representation to users), and the SC (to trigger response); they are also stored for offline processing and forensics purposes.

IV. Detection and Mitigation of Amplification Attacks

We use the ASTRID framework to detect and mitigate amplification attacks in an NFV service. We already discussed in Section II that the best mitigation strategy for DDoS attacks consists in blocking them at their origin, before they get amplified by vulnerable servers in the Internet. In this respect, a very common scenario is to block malicious flows at the boundary of a 5G network. Indeed, the advent of the Internet of Things (IoT) is expected to drastically increase the number of connected devices to the Internet. Coupled with their typical weak security posture, this makes them the ideal target for attackers that aim at building huge botnets. In this Section we briefly review the layout of a virtualized 5G service and describe the ASTRID components to implement DDoS protection.

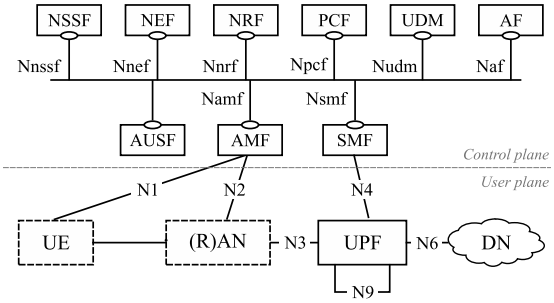

Fig. 4. ETSI architecture for the 5G core (5GC). The UE and AN are shown for reference, but they are not part of the 5GC.

A. Virtualized 5G Service

The 5G architecture maintains the distinction between the Access Network (AN) and the 5G Core (5GC), similar to previous models [9]. However, the new standard reflects the main advances in software-defined networking and moves to a cloud-native approach for the 5GC, where each functional entity is now modeled as a (virtual) Network Function (NF). This evolution is not limited to the nomenclature and the delivery of software instances instead of hardware appliances, but it also encompasses a sharper distinction between the control and user plane and a large transition from reference points (N[0-9]) towards service-based interfaces (*Nxxx*), at least in the control plane (see Fig. 4).

The service-based architecture facilitates both planning and management of the whole infrastructure. On the one hand, it allows more flexibility in placing NFs across centralized offices and computing/storage resources at the edge, with better support for latency-sensitive applications and user mobility. On the other hand, multiple NF instances can be deployed to create different network "slices," dedicated to vertical applications or user groups.

The 5GC architecture facilitates its implementation as NFV service, and indeed there are already both commercial and open-source implementations available. For our work, we selected the Open5GS project,[4] which can be deployed in conventional servers, virtual machines, and even Docker containers; specifically, Kubernetes manifest files are available, which makes it a perfect example for demonstrating the ASTRID framework.

For what concerns DDoS protection, the relevant part of the 5GC is the user plane, and in particular the User Plane Function (UPF), which bears the user traffic. In the 5GC architecture, the user plane carries Packet Data Units (PDUs) – Ethernet, IPv4 or IPv6 frames – from the User Equipment (UE) to an external Data Network (DN), and viceversa. The UPF is therefore the outermost NF in the 5GC that processes IP packets generated from or intended to the UE; it is mostly conceived for packet steering to/from multiple data networks, for anchoring in case of mobility, and for QoS and traffic policies enforcement.

[4]https://open5gs.org/

B. Threat Model

We address DDoS attacks that can be detected at their origin (namely, at the outermost edge of our 5GC) by anomalies in protocol usage. The 5G core indirectly bears the attack, but it is neither its origin, nor target. Therefore, in this paper we assume the 5GC a trusted and safe infrastructure to deploy our monitoring probes.

DDoS attacks are composed of a huge number of many small "rivulets" generated by compromised devices in a botnet, usually scattered over a large area. In these conditions, the volume of traffic at each site cannot be used as Indicator of Compromise in a reliable way, because it may be confused with periodical fluctuations. We therefore focus on detecting usage patterns that are known to trigger amplification. In our use case, we specifically analyzed two protocols, namely NTP and DNS. They are representative of the largest amplification factors and number of vulnerable servers, respectively. The simpler form of detection only considers anomalies in a single UPF, but correlation of data from multiple UPFs is also possible to spot the most distributed attacks.

The Network Time Protocol (NTP) has one of the largest amplification factors and a huge number of public servers; indeed, NTP is among the top emerging network attack vectors.[5] The NTP attack is based on sending a command called *monlist* to an NTP server; the server returns the addresses of up to the last 600 machines that it has interacted with. The request packet is only 234 bytes long, but the response may sum up to several dozens of kilobytes, depending on the number of returned addresses.

The Domain Name System (DNS) is one of the main pillars of the Internet, with a huge number of public and private servers deployed worldwide. Indeed, according to recent reports from cybersecurity vendors, amplification attacks account for more than one third of DNS issues experienced by enterprises [5]. These attacks aim at triggering the largest volume of DNS data in the response, usually by sending a query for "ANY" record, which retrieves all the available types for a given name. Alternatively, to elude security controls by some providers that already deprecated this meta-query, "SOA" or other types or record may be requested, but with a lower amplification factor.

C. Security Agents and Enrichment

According to the general description of the ASTRID framework given in Section III, the first step to define a SAP for DDoS detection is the enrichment of the NFV service with the necessary monitoring and inspection agents. In our use case, the UPF is the only function which processes user traffic, so security agents must be placed there (see Fig. 5). UPFs are software functions that can be easily replicated for scalability, therefore they probably represent the only point in the 5GC (and also the Internet) where packets can be inspected with fine granularity. Indeed, the far edge of a 5G architecture is the RAN, which would represent an even better location for inspection and enforcement; it would also lead to more scalability, because of the lower amount of aggregated traffic. However, this part is not included in the 5GC implementation selected for our work, because this element can be shared by multiple network instances operated by different telco providers.

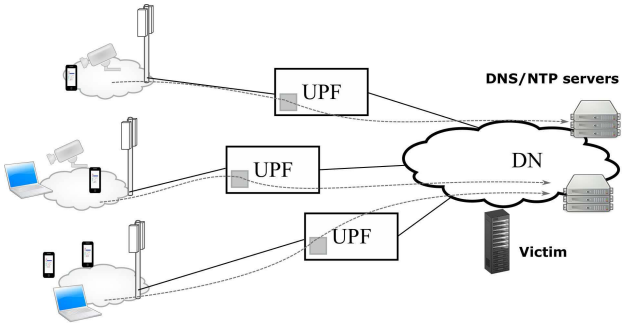

Fig. 5. Security agents are placed at the far edge of the 5GC.

There are several tools that can be used to monitor network flows and inspect packets. However, they are usually designed to run in physical servers or Virtual Machines, and do not fit well PaaS and serverless environments. Furthermore, in the telecommunication domains there is still a prevailing attitude towards proprietary and closed-source solutions, which are usually delivered as immutable and certified software images; this is mostly necessary to preserve their integrity and to ensure telco-grade performance are not compromised by third party's software.

To effectively address ever-evolving attack patterns without breaking the immutable infrastructure mindset, we pursue an agentless approach that instruments the underlying kernel without the need to change the container images. In this respect, we leverage the extended Berkeley Packet Filter (eBPF), a Linux native technology for instrumenting the kernel at run-time. eBPF programs follow the same development process of userspace applications and can be created independently from the kernel development process. They are dynamically injected into the kernel, hence they allow the creation of inspection tasks that are tailored to the actual requirements of external security analytics, as opposed to static kernel implementations. The kernel provides a run-time verifier that analyzes all possible execution branches of an eBPF program before loading it. eBPF programs are rather limited in terms of memory access and available functions; indeed, operations outside the eBPF environment are carried out only through specific "helper" functions that further limit the likelihood of introducing faulty programs.

One major limitation of eBPF programs is that they require a specific userland counterpart to take care of control and management. For example, Suricata is currently able to load custom eBPF programs, but they are limited to a few operations (filtering, bypass, load balancing) and must comply with specific programming patterns.[6]

[5]See https://blog.cloudflare.com/network-layer-ddos-attack-trends-for-q3-2020.

[6]https://suricata.readthedocs.io/en/latest/capture-hardware/ebpf-xdp.html#setup-ebpf-bypass

We addressed this issue by using *Dynmon*,[7] a transparent service that allows the dynamic injection of eBPF code in the Linux kernel, enabling the monitoring of the network traffic and the collection and exportation of custom metrics. It is part of Polycube, a framework conceived for developers of network functions in the NFV world [10]. If a NS is built with Polycube, it is already a fully agentless system; otherwise, Polycube can be deployed as additional security sidecar, as in our use case.

The remarkable aspect of our approach is that exactly the same userland utility (namely, the Dynmon service) is used to load and run programs that generate different metrics, without specific constraints on the data structure.[8] A large number of eBPF map types is currently supported, including plain and per-cpu hash, LRU hash, array, queue and stack. Finally, it also supports atomic eBPF maps content read thanks to an advanced map swap technique, and maps content deletion when read.

The possibility to run custom code brings unprecedented flexibility in defining inspection rules, which are no more bounded to pre-defined patterns, as it happens in signature-based detection tools as Suricata, or events, as in Zeek, and are also more efficient than regular expressions or similar logic. In fact, the Dynmon programming model is looser than what required by Suricata, allowing a larger degree of extensibility (even if the scope of the two tools is rather different). Both Suricata and Zeek give access to a large number of protocol fields;[9] the Zeek script language is far more powerful than Suricata rules, but they are interpreted at run-time and not suitable for high packet rates.

We developed three eBPF programs for each protocol in our use case with the following functionalities:

- coarse-grained measurement of the total number of packets bearing DNS/NTP data;
- fine-grained deep packet inspection that counts the packets with potentially harmful requests (i.e., the *monlist* command for NTP and "ANY" query for DNS);
- fine-grained measures and dropping of packets that carry the potentially harmful requests.

The first type of programs gives a raw indication of anomalous conditions, which may be due to attacks or deviations from common usage patterns. The second type of programs is more suited for the purpose of detecting the amplification attack; however, it implies deep packet inspection and that usually slows processing down.[10] The last type of programs is used for enforcement: in our implementation we drop all potentially harmful requests, based on the consideration that they are not usually necessary for licit operation and this behavior is already implemented by some providers.[11]

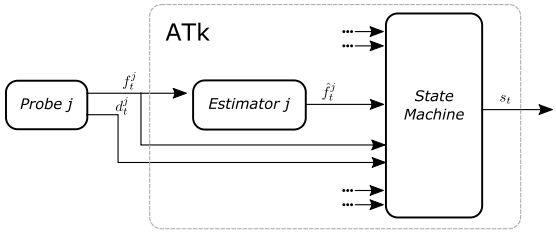

Fig. 6. Architecture for security analytics.

D. Context Manager and eBPF Abstraction

The eBPF programs described in Section IV-C are modeled as standalone agents in the Context Manager. Differently from conventional agents (e.g., Elastic beats) that must necessarily be instantiated at deployment time, eBPF programs can be loaded at runtime through the Dynmon service without breaking service continuity. This means that additional programs can be developed and loaded to inspect other vulnerable protocols beyond DNS and NTP, without any change to the running instance of the 5GC.

The model for an eBPF agent includes its source code and the description of the metrics it provides. The source code is necessary because the program must be compiled for each specific environment by Dynmon before being loaded into the kernel. It is also used to parse maps structures and to instantiate them. The description of the metrics is again used by Dynmon to map eBPF maps to OpenMetrics names and metadata; this simplifies the necessary parsing inside security analytics.

E. Analytics ToolKit

We implemented a general-purpose Analytics ToolKit (ATk) that can be applied to detect a plurality of attacks. In the following, for the sake of brevity, we only refer to the DNS use case, for which we report numerical evaluation in Section V, but we have fully implemented and tested the necessary software for NTP amplification attacks too.

The ATk analyzes quasi real-time data, performing estimations based on historical time-series, and finds anomalies in the input pattern. The architecture of the ATk is shown in Fig. 6. It aggregates the data published asynchronously by each monitoring agent j (*probe*) located in the set of network nodes $\mathcal{N}$ at each time instant t: coarse grained measurements (namely, volume of DNS packets) denoted by f_t^j, and fine-grained measurements (namely, volume of "ANY" query) denoted by d_t^j. In general, f_t^j and d_t^j can be arrays, though in this use-case they are scalars. Nodes are grouped into a set of A areas $\mathcal{A}_i$, based on their geographical location ($\mathcal{A}_i \subseteq \mathcal{N}, \quad \bigcup_i \mathcal{A}_i = \mathcal{N} \ \ \forall i = 1, \ldots, A$).

Internally, a *proxy* service (not shown in the schematic architecture of Fig. 6) consumes the measurements and dispatches

[7] https://polycube-network.readthedocs.io/en/v0.9.0-rc/services/pcn-dynmon/dynmon.html

[8] The main limitation is that *structs* and *unions* types are not supported. This is mostly due to the export formats (JSON and OpenMetrics), because OpenMetrics does not support complex data structures.

[9] See, for instance, the list of available keywords for Suricata rules: https://suricata.readthedocs.io/en/latest/rules/index.html, and the list of protocol analyzers for Zeek: https://docs.zeek.org/en/master/script-reference/proto-analyzers.html#zeek-dns.

[10] The number of DNS/NTP packets can be easily computed by hardware appliances based on the destination port of UDP packets. The detection of *monlist* packets and "ANY" queries requires a software approach, since it cannot be generalized for any protocol.

[11] https://blog.cloudflare.com/deprecating-dns-any-meta-query-type/

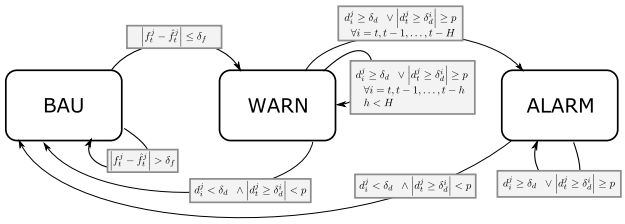

Fig. 7. ATk state diagram.

them to a battery of $N = |\mathcal{N}|$ *estimators*, each one processing the data from a different node. In this architecture, the number of *estimators*, which are the most computationally heavy functions, can scale horizontally with the number of nodes and/or use-cases to keep the ATk responsive without consuming unnecessary resources. From a practical perspective, a small group of *estimators* (nine in our setup) is deployed in a separate Kubernetes pod. This ensures scalability of the ATk as the number of UPF nodes and measures increases. The *proxy* is also located in a separate pod; it represents a potential bottleneck for the system, but our experiments showed that it introduces a negligible load and latency for the considered use-cases.

Each *estimator j* is responsible to predict the value $\hat{f}_t^j$ expected at time *t*, by using the previous *T* values (from $t-1$ downto $t-T$). Their implementation use standard ARIMA (AutoRegressive Integrated Moving Average) models. The larger *T*, the more daily and weekly patterns are captured by the *estimators*. ARIMA was selected because it can be continuously trained during operation, as explained in detail in the following, hence removing the need for large seasonal datasets.

Both measures and predictions from the *N* estimators drive the ATk state machine, which is shown in Fig. 7. There are three states, corresponding to normal (*BAU*), suspicious (*WARN*) and attack conditions (*ALARM*). The state machine is re-evaluated every Δt, by comparing the current measurement f_t^j with the estimation $\hat{f}_t^j$ computed from the previous *T* values. Both the current status and correlations (see below) are collected by the *proxy* and published on the Kafka bus.

The ATk persists in *BAU* as long as $|(f_t^j - \hat{f}_t^j)/\hat{f}_t^j| \leq \delta_f$, where δ_f is the *forecast* threshold (calculated as percentage of the forecast value), otherwise it switches to the *WARN* state. To start the process, a time-series with *T* "clean" measurements must be available, i.e., coming from nodes which are not under attack, in order to learn the periodic behavior. During operation, the new measurements f_t^j are continuously added to the time-series to continue training; however, real values are replaced by their estimation $\hat{f}_t^j$ when the state is not *BAU*, to prevent overfitting.

In the *WARN* state, fine-grained measurements d_t^j are also collected to improve the accuracy of the detection. These values are compared with two fixed *detection* thresholds δ_d and δ_d^i that account for individual deviations and correlated variations in area $\mathcal{A}_i$, respectively. The ATk switches and remains in the *ALARM* state when either one node exceeds the threshold δ_d or *p* nodes in the same area $\mathcal{A}_i$ exceed the threshold δ_d^i, in both cases this should happen for H consecutive evaluations. This can be formally described as:

$$\begin{aligned} &\exists j \in \mathcal{N} : d_i^j \geq \delta_d \quad \vee \\ &\exists \mathcal{A}_i \ (i = 1, \ldots, A) : \left| \{ j \in \mathcal{A}_i \} : d_t^j \geq \delta_d^i \right| \geq p \\ &\forall i = t, t-1, \ldots, t-H \end{aligned} \tag{1}$$

The first equation allows to identify attacks that go through a single node; the second equation is more suitable for very distributed attacks, that affect a plurality of nodes of the same area with smaller deviations. All performance evaluations presented in this paper have been made considering attacks through a single node: detection performance in case of correlated variations are left for further study.

F. Response and Mitigation

In order to respond to attacks, a number of response policies in the form of Drool rules are loaded in the SC as part of the SAP. These rules define specific actions to be triggered for each state of the ATk (BAU, WARN, ALARM). The structure of the rules is basically the same for each state, and includes the following list of actions:

- previous eBPF programs are removed via CM;
- the eBPF program corresponding to the current state is loaded through the CB.

The different eBPF programs have been already described in Section IV-C. Specifically, the program for coarse-grained measures is loaded in BAU state, the program for fine-grained measure is loaded in WARN state, and the program for enforcement is loaded in the ALARM state.

G. Security Analytics Pipeline

According to the description of the ASTRID framework, a SAP must be setup for bearing data from monitoring agents to the detection algorithm. For the specific case of amplification attacks, the ASTRID sidecar includes a Logstash agent that periodically polls the Dynmon service in Polycube to get the current measurements (i.e., total number of DNS packets and number of "ANY" queries seen, when available). The same agent publishes this data on the Kafka bus.

The SC configures a common Kafka topic for both Logstash and the ATk when the SAP is instantiated. This avoids the need for filtering relevant data at the ATk, and also prevents flooding other detection algorithms that might be running on the same platform. In the ATk, a *proxy* is then responsible to deliver data from each UPF to the corresponding estimator, as described in Section IV-E.

V. Numerical Evaluation

We carried out numerical evaluation of the proposed framework. The analysis focused on the main innovations aspects of our approach, namely *i*) inspection of network packets with eBPF programs; *ii*) anomaly detection via network analytics implemented in the ATk; and *iii*) response time to start mitigation when an alarm is received.

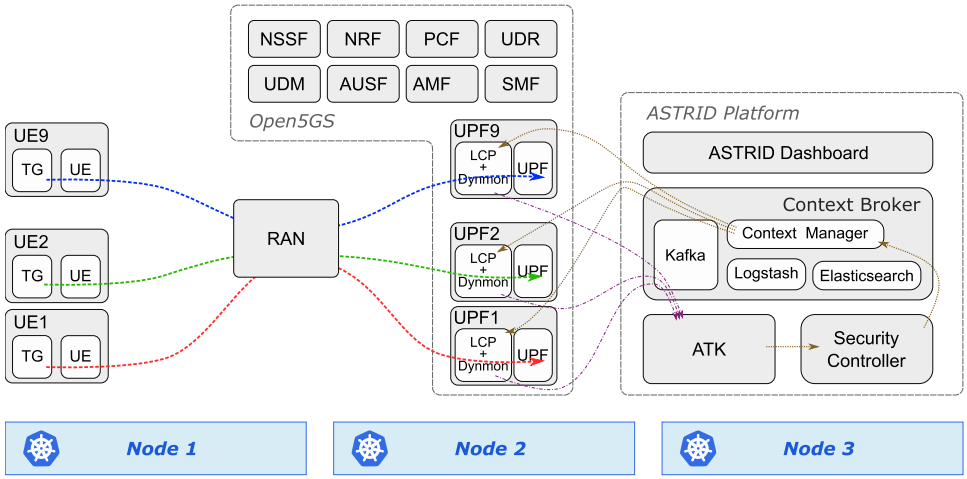

Fig. 8. Experimental setup of a virtualized 5GC monitored by ASTRID.

A. Experimental Testbed

For the purpose of evaluation, we set up an experimental testbed that replicates all the elements present in a 5G installation. Fig. 8 shows the structure of our testbed. It is composed of the ASTRID framework,[12] Open5GS service, RAN emulation. Grayed boxes correspond to Kubernetes pods, and white boxes to internal containers. The dashed boxes group pods that logically belong to the same service.

The 5G network is made of 9 UPFs and a common control plane. Each UPF has its own standalone Data Network (which is not connected to any other component), hence resembling the typical structure for edge computing. In addition to the 5GC, an access network is also emulated by software (hence without real hardware and radio transmission); the necessary components are taken from the UERANSIM project[13] and include both a gNodeB (namely, the virtual RAN) and a virtual UE.

Traffic Generators (TG) are included in the UE to emulate local applications or IoT devices that make traffic towards the 5G network. They have been implemented with the Python Scapy library that enables the user to send, sniff and dissect and forge network packets. Scapy allows to define a set of packet templates, to send them, and to receive the corresponding responses. It also allows to match requests with responses and returns both a list of packet pairs (request, response) and a list of unmatched packets. The main advantage with respect to other tools like *nmap* or *hping* is that Scapy returns the whole packet, instead of the plain indication of open/closed/filtered port.

A single RAN gives connectivity to 9 UEs. These nodes are grouped into 3 non-overlapping areas. In our configuration, each UE-X generates traffic (DNS/NTP packets) towards the corresponding UPF-X. This is achieved by associating each user to a different DN, and each DN to a different UPF in the SMF. We argue that this structure is not representative of a realistic setup, because usually at most one UPF is associated to each RAN. However, we kept the setup simple for those elements that are not involved in the evaluation of our monitoring and detection mechanisms. All protocols and tunnels envisioned by the 5G architecture are used to deliver the traffic from the TGs to the UPFs.

The TGs have been effectively used to generate synthetic attack data applying the network traffic profiles disclosed and analyzed in the literature [11], [12] and verified by a three-months test campaign conducted by Ericsson on field involving a customer telco operator. These profiles follow a sinusoidal pattern, which is representative of typical daily variations. The ATk is always initially trained with this profile, but additional variance is added during the measurements to account for inaccurate data and short-term variations.

The testbed is deployed in a 3-nodes Kubernetes clusters. Each node has 2 Intel Xeon CPU E5-2660 v4@2.00GHz with 14 cores and hyperthreading enabled, 128 GB RAM, 64 GB SSD storage; they are interconnected by a Gigabit Ethernet network. Demonstration of this setup is available as external material.[14]

B. Enrichment and Deployment

The transition to NFV is largely motivated by more agility and reduced development times for NSs. In this respect, security processes must not slow down the main devops workflow, and this directly applies to the design and deployment of SAPs in our framework.

The time to design a pipeline depends on the complexity of the pipeline, the number of agents that need to be inserted and configured, and the user expertise. The dashboard provides a graphical pipeline editor that allows to easily add and configure agents to the service graph by pointing and clicking within a couple of seconds. Moreover, in case of large graphs where the same agent(s) must be deployed in many components, the enrichment process can be executed programmatically by providing code to rewrite and enrich the original manifest file.

The time to deploy a NS with Kubernetes depends on several factors. This includes, for instance, the size of the graph,

[12] https://github.com/astrid-project/astrid-framework
[13] https://github.com/aligungr/UERANSIM
[14] https://www.youtube.com/watch?v=ZdRkUz6yzyY

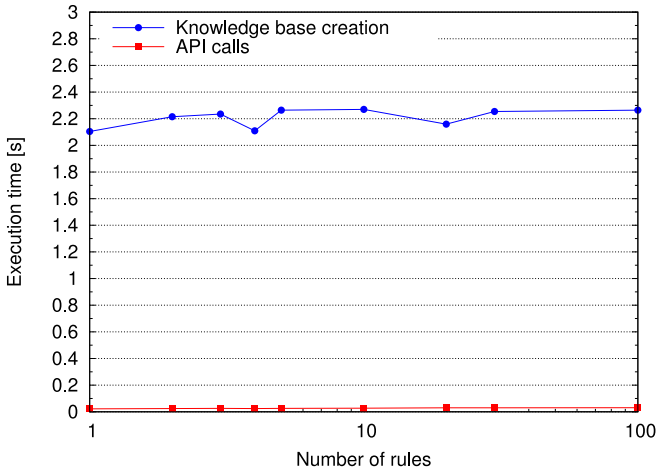

Fig. 9. Breakdown of the latency of the SC.

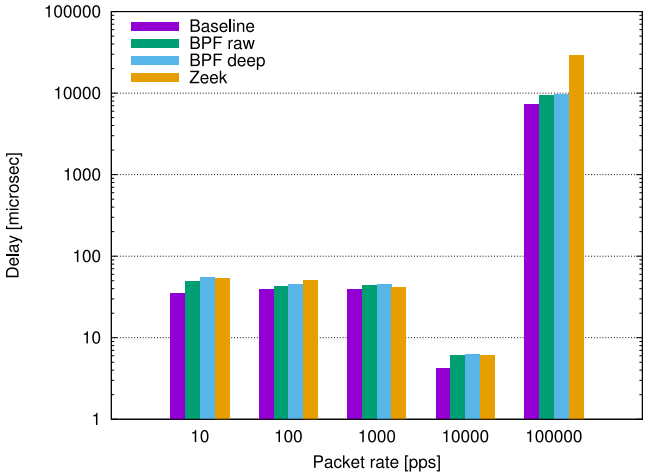

Fig. 10. Latency experienced by packets across the UPF, while increasing the packet rate. Log scale is used for the y axis.

the availability of container images in the local caches, and the current workload of the Kubernetes node(s). However, there is no notable difference between a deployment with or without security agents in our case and, therefore, the impact of the enrichment process can be neglected.

After deployment, the SC is expected to configure the SAP. This includes i) the creation of the corresponding knowledge base (namely, rules, processes, functions, type models), ii) execution time of API calls. Fig. 9 reports the measures for different number of rules in the policy database, starting with 1 rule execution and until 100 event message execution in order to detect attacks. As it can be seen from the graph, the most time consuming operation is the creation of the knowledge base. In fact, 2 seconds are required to initiate the knowledge base for 100 rules. Instead API calls to other components is a less time consuming task, in the order of milliseconds, which does not increase with the number of rules.

C. Inspection

The implementation of DPI raises the issue of scalability in case of high traffic load. The target is to achieve the same performance of the "baseline" scenario, namely when no inspection process is run, for the traffic volume forwardable in a pure software environment. This is enough for most use cases, as higher traffic volumes are usually managed by multiple UPF instances. Since we do not expect meaningful differences in the behavior of our system with different protocols, we only investigate DNS in our experiments.

To evaluate performance of the security agents, we considered both their impact on packet processing (e.g., delay introduced in network flows) and resource consumption in terms of CPU/memory usage. This answers the question whether the inspection mechanism is really transparent to network operation and PaaS deployment.

We also demonstrate the advance with respect to existing technology by comparison with a well-known and widespread network monitoring tool, namely Zeek. The choice was motivated by the fact that this tool performs a similar role than ASTRID agents, namely it provides raw measurements and delegates the detection to external processes. Zeek parses network packets and generates events, which contain relevant fields extracted from the messages and are then processed by a powerful scripting language. We therefore developed a simple script that counts the total number of DNS requests and the number of occurrences with a query type of *ANY* (all relevant packet fields are always included in the corresponding event).

To perform extensive tests that are not affected by other parts of the setup (namely, the UE and RAN), we directly streamed DNS packets through a single UPF, hence bypassing the access network. In our experiments we replicated two kinds of real-word DNS packets: a simple query for an A-type record (which requests the translation of a name into an IPv4 address), and a potentially rogue query of type ANY. To get meaningful results, each test was run for at least 30 seconds (tests with lower packet rate were run longer to get a larger number of samples). Our evaluation considered variable packet rates of 10, 100, 1000, 10,000 and 100,000 packets per second.

1) Impact on Network Traffic: The first evaluation considers the impact of the inspection agents on IP traffic. Our main finding was that we were able to receive packets at the full transmission rate, without packet loss. We only observed 0.42% lost packets for Zeek with the highest transmission rate. We therefore conclude that either inspection technologies does not affect packet transmission.

2) Forwarding Delay: Even in the absence of packet loss, the deployment of an inspection probe may increase the processing time, which turns into larger forwarding latency. We therefore considered the overall latency experienced by packets across the UPF. To this aim, we measured the difference in the timestamps recorder by *tcpdump* for each packet on the ingress and egress interface.

Fig. 10 compares the latency introduced by our programs and Zeek with a baseline scenario; the latter corresponds to the situation when no probe is applied. For our eBPF programs, "raw" indicates the simple packet counter and "deep" indicates the counter of packets with the ANY query.

With lower packet rates (namely, below 10,000 packets per second), the impact of the considered inspection agents is

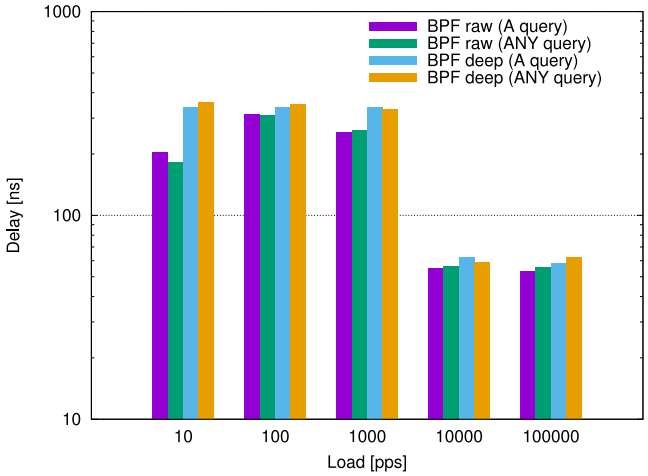

Fig. 11. Execution times for eBPF programs, when processing A and ANY queries. The log scale is used.

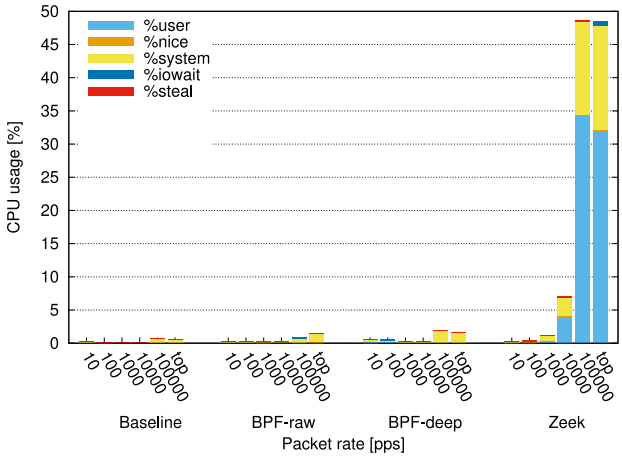

Fig. 12. Cumulative CPU in the UPF, while varying the transmission rate of DNS packets.

rather low. Indeed, even if the logarithmic scale largely hides the effect, the agents introduce up to 50% additional latency with respect to the baseline scenario. However, the overall latency is always below 100 μs, which is anyway more than acceptable for Internet traffic.

In some cases, Zeek seems to perform better than eBPF programs. However, while eBPF programs are really extensions to the operation of the in-kernel network stack, Zeek captures packets with a raw socket and then the inspection happens in parallel to forwarding operations implemented by the kernel. Overall, achieving close results to Zeek means that the impact on standard kernel operations is rather low. In addition, we remark that results may slightly vary for different realizations of the experiment, hence we can consider the performance almost equivalent.

Rather unexpectedly, the latency was lower at the rate of 10,000 packets per second. We repeated the experiments several times, and got the same results. One possible reason is better efficiency with this rate in the reception process (namely, fewer hardware interrupts are generated); in any case, since our objective is only to compare the tools, we did not investigate in detail the cause of this behavior.

With the largest packet rate, the gain with respect to Zeek largely increases. In this case, the latency introduced by Zeek almost doubles with respect to eBPF programs (note that Fig. 10 uses the log scale). Overall, the difference between raw and deep inspection is rather limited.

3) Processing Delay: We already pointed out in the previous Section that eBPF programs are de-facto extensions to the operation of the kernel. So it is interesting to look at the time to execute such programs. Even if this measurement is not trivial to perform, we provide a reasonable estimation by measuring the elapsed time from within the same program (namely, timestamps are taken at the entry/exit point). Of course, this does not account for the overhead to invoke and run the eBPF function.

Fig. 11 shows the execution times measured under different conditions. This time, we also considered the potential impact of different types of queries, namely A and ANY. We note the impact of the different types of programs, namely for raw and deep packet inspection. The latter requires more time, as expected, because of the larger number of operations to be performed to parse the packet. However, the differences when measuring or inspecting A and ANY queries can be only ascribed to normal variability of different realizations. Overall, the processing time is always well below 1 μs, hence confirming the high efficiency of this inspection method.

Similarly to the previous experiment, shorter processing times are measured for the highest packet rates. Again, the same behavior was observed for different realizations and we do not have an explanation for this behavior beyond the intuition of a better management of interrupts; however, it is worth being deeper investigated in the future.

4) Resource Usage: Finally, we consider resource usage, by investigate CPU utilization and memory allocation.

Fig. 12 shows the cumulative CPU usage for the different types of eBPF programs and Zeek, broken down into the usual components reported by common CPU monitoring tools. In this case, in addition to the packet rates already used for previous experiments, we also considered the fastest rate we were able to send packets (which is almost the same that can be achieved with 100,000 pps), indicated as "top." Even if the figures cannot be ascribed to the monitoring tools only, it is clear that the impact of Zeek on CPU usage is more than an order of magniture greater than Polycube. In the case of Polycube, we note an increment of system CPU usage with higher transmission rates, due to eBPF operation in kernel-space, whereas there is no dependency for the user space, because Dynmon only collects data with the same rate. Instead, Zeek splits operation between kernel-space (for capturing and duplicating packets) and user-space (for packet inspection), and this is reflected in the increment of both shares of CPU usage. The computing overhead of Zeek rises up from 10,000 packets onward, which are the likelihood conditions for a large-scale attack.

Finally, Fig. 13 shows an analysis of memory allocation. We consider the Virtual Memory Size (VMS), the Resident Set Size (RSS), the Proportional Set Size (PSS), and the

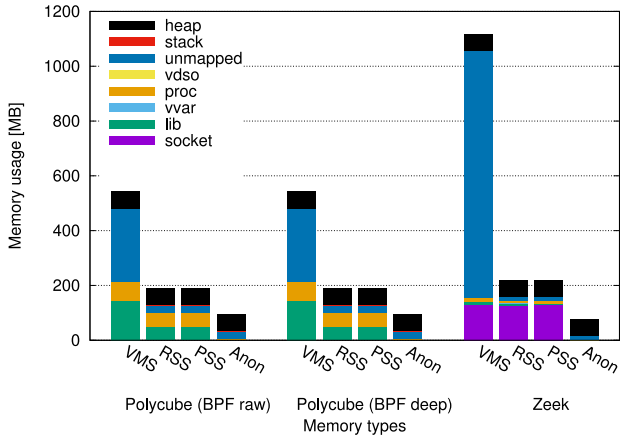

Fig. 13. Memory allocation for the different tools.

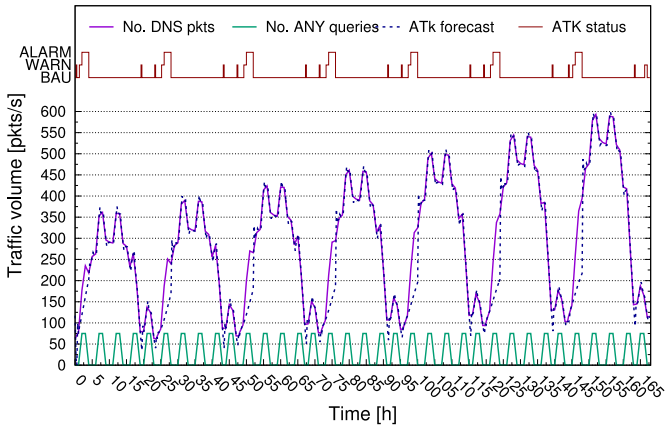

Fig. 14. ATk status during one-week simulation, with increasing traffic profile.

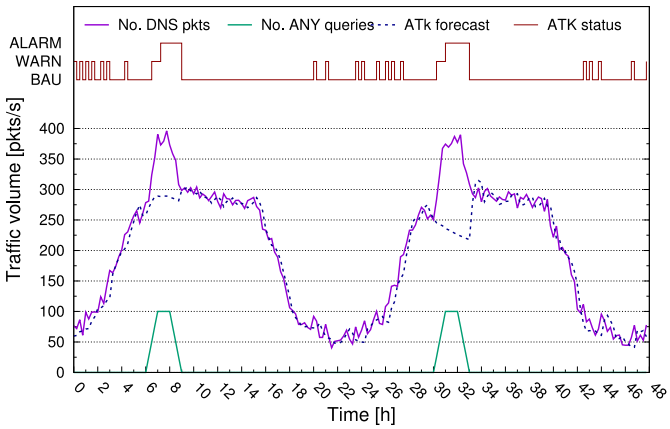

Fig. 15. ATk status during a two-day simulation, with a single attack during peak hours.

Anonymous share (Anon). The first corresponds to the overall address space allocated by the program, the second is the real size of physical memory allocated (including shared libraries), the third is again the physical memory but with proportional attribution of shared libraries (namely, their memory allocation is divided by the number of programs that use them), and the forth is memory mappings not backed by a file. The amount of memory is further broken down according to the corresponding mapping type (e.g., file, library, stack, heap, process, socket, etc.), to give a more accurate picture of memory usage. Not surprisingly, the VMS of Zeek is far larger than Polycube, because the program is in general more complex. It is also rather intuitive that the eBPF programs does not impact memory allocation for Polycube. Also the RSS and PSS of Zeek are higher than Polycube, even if proportionally less than the VMS. Indeed, RSS and PSS are almost equivalent in our setup, because libraries are not shared with other processes inside the docker container. Zeek allocates less memory to the process itself and shared libraries ("proc" and "lib" slices, respectively) than Polycube, but it has around the same heap size. Remarkably, a large slice of memory indicated as "socket" is used for capturing packets.

D. Detection

For what concerns attack detection, we investigated the accuracy of the ATk and how it behaves in different conditions. For the sake of simplicity, we focus our analysis on a single UPF, by generating a variable traffic profile, as described in Section V-A. The internal estimators use ARIMA (5, 0, 0). The volume of DNS messages is always referred to the measurements from the probes, hence it is expressed as number of packets in the polling interval, which is set to 15 minutes. The thresholds are set to $\delta_f = 10$, $\delta_d = 50$ (packets/interval), unless differently indicated.

First of all, we tested the capacity of the ATk to correctly follow a profile that changes after the initial training. For this purpose, we initially trained the ATk with a traffic profile ranging from 50 to 300 packets/interval. Then, we gradually amplified the traffic profile up to 550 packets/interval during one week, to emulate a sort of seasonal periodicity (which was not used in the training phase). We also generated 5 small attacks (75 packets/interval, corresponding to the peaks of the fine-grained measurements) that could potentially deceive the learning process. Despite of the adverse conditions, Fig. 14 shows that the ATk continuously learns the traffic profile in a correct way, and its estimations accurately follow the real measurements. Indeed, neither the increasingly volume of DNS packets nor the attacks trigger false positives. Unfortunately, in these conditions, the ATk largely fails the detection, because of relatively small size of the attack.

Fig. 15 shows the evolution of the ATk during two days of simulation. This time, a single larger attack is generated during peak hours (100 packets/interval); this is the most challenging situation, because the relative size of the attack with respect to licit traffic is smaller than in other hours. To make the evaluation more realistic, we introduced some variance (10 packets/interval, Gaussian distribution) to the traffic profile used for training. The ATk correctly detects the attack, by switching to the WARN state first, and then going to in ALARM. Of course, due to the transition through the WARN state, the detection is delayed. To mitigate this latency, measures can be collected more frequently, since their transmission has a negligible overhead. Notably, there are a lot of transitions to the WARN state, due to the inaccuracy of

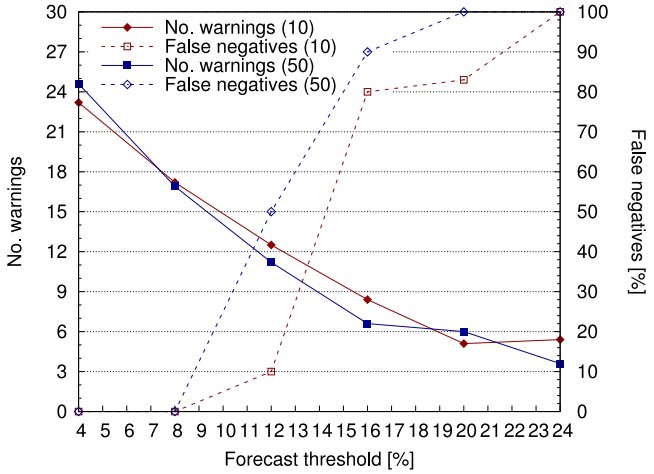

Fig. 16. Tradeoff between inspection overhead and detection accuracy, while varying the forecast threshold δ_f.

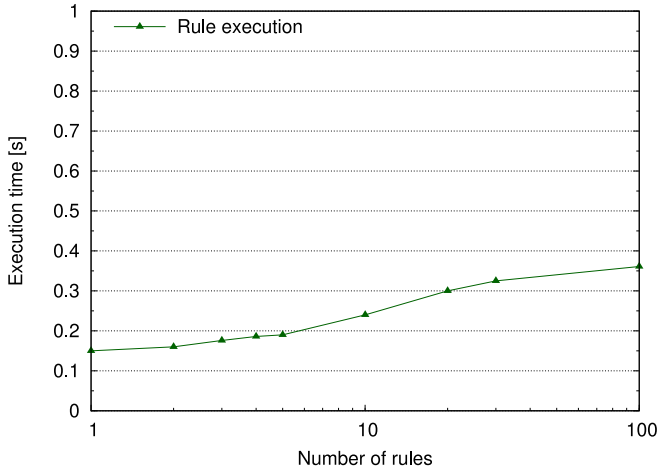

Fig. 17. Execution time of SC's rules.

using coarse grained measurements only. These transitions have a cost for the system, because they activate DPI in the probes (see Section V-C), hence it is important to reduce their number.

We therefore investigated the tradeoff between inspection overhead and detection accuracy, by carrying on a sensitivity analysis to the value of the forecast threshold. A single attack was again simulated during one day. Its volume was about 1/3 of the maximum daily volume of licit traffic and placed in the most unfavorable time, namely during the peak hours. All simulations were repeated ten times, and again a variance of 10 packets/interval was added to the original training profile. We measured the number of "warnings," which entail changes in the eBPF code, and lead to instability of the ATk status. Based on ATk state machine described in Section IV-E, the minimum possible number of warnings is equal to the hysteresis between WARN and ALARM states, which is 3.

Fig. 16 shows the results from the experiments. We considered different values of the forecast threshold, expressed as percentage with respect to the maximum size of the attack (this way we can set the absolute value of the threshold based on the attack size that we can tolerate), and two values for the detection thresholds (10 and 50 packets/interval). Overall, the number of transitions to the WARN state ranges from 25 for $\delta_f = 4\%$ down to a minimum of 5 for $\delta_f = 24\%$. It smoothly decreases as the forecast threshold increases, but it does not depend on the detection threshold because the commutation between BAU and WARN is only governed by the forecast threshold. Correspondingly, the number of False Negatives increases for larger values of the detection threshold; it is systematically higher for larger values of the detection threshold, as expected. From this analysis, we can select the best forecast threshold based on the maximum number of False Negatives that we are willing to tolerate (e.g., 5%). In general, it would be convenient to select the lower detection threshold, because it corresponds to minimal number of published warnings. Finally, we argue that unfortunately the detection threshold cannot be arbitrarily low, to avoid the detection of False Positives.

TABLE I
BREAKDOWN OF THE DELAY (IN SECONDS) TO PUSH AN EBPF PROGRAM TO THE NETWORK AGENT

Operation	CM	Polycube	Total
Add coarse-grained meas.	0.0221	2.3792	2.4013
Remove coarse-grained meas.	0.1628	0.0296	0.1924
Add fine-grained meas.	0.0502	2.4033	2.4535
Remove fine-grained meas.	0.1726	0.0302	0.2028

E. Response and Mitigation

In the response phase, events generated by the ATk triggers rules in the SC. The execution time of the Drools engine is directly proportional to the number of rules in the policy, as shown in Fig. 17. For the testbed considered in this paper, the number of rules in the database does not exceed 20. Hence, it requires on average 0.2 seconds to start response actions. For instance, to switch between BAU, WARN, and ALARM states it takes 0.192 seconds.

However, the time required to finalize the response depends on other components that are involved in the action. We therefore investigated in details the composition of this delay, by splitting it into two components: processing introduced by the CM and configuration of the local agent (Polycube). Table I shows that the delay is rather limited, around a couple of seconds, and have a marginal impact on the timescale of the ATk. Overall, the major impact is due to loading the eBPF program into the kernel, because this operation implies compilation and code verification. Oddly, the CM is slower in the removal process; this is due to the internal implementation that stores the current configuration in Elasticsearch and needs more time to remove an element.

VI. RELATED WORK

After the amplification DDoS attacks hit the headlines around 10 years ago, several researchers have started investigating the pattern and impact over the global Internet [12], [13]. Due to the difficulty to monitor the public Internet, the interest has often focused on the prevention

of these attacks [14]. However, some authors tried to identify relevant features in historical data [15].

More recently, the advent of Software-Defined Networking has revived the interest in detection, especially by applying machine learning techniques that use generic network features like the size of packets, duration of the connection, etc. [16]–[20]. Since the training is not trivial with real traffic, reinforcement learning was also investigated by authors [21] that observes the pattern directly from the traffic. Some authors also explicitly considered the geographical dimension of the problem [22]. However, we argue that SDN is not capillary used in practice, especially in larger telco's networks. Overall, we argue that SDN cyberdefence applications provide a scalable and flexible solution for physical infrastructures, but cannot be seamlessly integrated with generic VNFs.

Several authors already proposed to leverage NFV to replace rigid and expensive hardware cybersecurity appliances with software instances [2], [23], [24]. Differently from our approach, these authors build standalone detection systems, which are not integrated with NSs. Additionally, they mostly focus on the implementation of the internal detection and mitigation mechanisms but does not address implementation issues and applicability to modern virtualization paradigms.

Beyond the analysis of network traffic, there are also authors that consider logs for the configuration of firewall rules [25]. It is important to note that SHIELD [26] also exploits NFV for adaptive monitoring of an IT infrastructure and for feeding the data to an analytics engine to detect attacks in real time. One of the key novelties of the ASTRID project in contrast to SHIELD is the use of eBPF modules. However, SHIELD provides the protection of the modules inside the framework.

VII. Conclusion

In this paper, we have described a framework to create security analytics over virtualized NFV services. Our approach enriches existing service templates with security agents and connects them to additional elements for data collection and attack detection and response. We applied this framework to a common use case, namely the detection of distributed amplification attacks in a virtualized 5G Core.

Our solution extends the original NS template with sidecar security agents, without the need to change service images; in addition, we leverage the eBPF framework to load inspection and enforcement programs in the running kernel, which does not disrupt service operation. By analyzing data coming from the edge of the 5G Core, we can detect anomalies with respect to the expected behavior; we also perform continuous learning so to follow changes in usage patterns.

Performance analysis has shown that packet inspection has negligible impact on forwarding operations, and that our implementation is more efficient than existing general-purpose tools for collecting specific measures. However, further investigation is needed to understand if this still holds with a larger number of measures. About the accuracy of the detection, we can correctly identify DDoS attacks before they get amplified, and our implementation is quite robust to periodic fluctuations.

Future work will consider better integration of the monitoring framework in PaaS environments, to become truly agentless. In addition, correlation of data from multiple areas will be considered, to detect even lower attack volumes.

References

[1] D. Montero *et al.*, "Virtualized security at the network edge: A user-centric approach," *IEEE Commun. Mag.*, vol. 53, no. 4, pp. 176–186, Apr. 2015.

[2] B. Rashidi, C. Fung, and E. Bertino, "A collaborative DDoS defence framework using network function virtualization," *IEEE Trans. Inf. Forensics Security*, vol. 12, pp. 2483–2497, 2017.

[3] A. Carrega *et al.*, "Situational awareness in virtual networks: The ASTRID approach," in *Proc. IEEE 7th Int. Conf. Cloud Netw. (CloudNet)*, Tokyo, Japan, Oct. 2018, pp. 1–6.

[4] M. Repetto *et al.*, "Leveraging the 5G architecture to mitigate amplification attacks," in *Proc. IEEE Int. Conf. Netw. Softwarization*, Tokyo, Japan, Jun. 2021, pp. 443–449.

[5] "The Trust Factor—Cybersecurity's Role in Sustaining Business Momentum; 2018–2019 Global Application & Network Security Report." Radware. 2019. [Online]. Available: https://blog.radware.com/wp-content/uploads/2019/09/ERT-Report-2018-2019.pdf

[6] "Quarterly DDoS and Application Attack Report." Radware, Oct. 2021. [Online]. Available: https://discover.radware.com/aaf9f19d-33da-5747-b631-97a2f1ba0f92

[7] "Cloud DDoS Protection Service: Attack Lifecycle Under the Hood; Technology Overview Whitepaper." Radware. 2016. [Online]. Available: https://www.radware.com/assets/0/314/6442477977/2c6454b4-403b-45b1-ac58-dc628bc210b3.pdf

[8] N. MacDonald and T. Croll, "Market guide for cloud workload protection platforms," Gartner, Stamford, CT, USA, Rep. G00716192, Apr. 2020.

[9] "5G; system architecture for the 5G system; version 15.3.0, release 15," 3GPP, Sophia Antipolis, France, Rep. ETSI TS 123 501, Sep. 2018.

[10] S. Miano, F. Risso, M. V. Bernal, M. Bertrone, and Y. Lu, "A framework for eBPF-based network functions in an era of microservices," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 133–151, Mar. 2021.

[11] R. Bolla, R. Bruschi, A. Cianfrani, and M. Listanti, "Enabling backbone networks to sleep," *IEEE Netw.*, vol. 25, no. 2, pp. 26–31, Mar./Apr. 2011.

[12] J. Czyz, M. G. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. D. Bailey, and M. Karir, "Taming the 800 pound gorilla: The rise and decline of NTP DDoS attacks," in *Proc. Conf. Internet Meas. Conf. (IMC)*, Vancouver, BC, Canada, Nov. 2014, pp. 435–448.

[13] A. Wang, W. Chang, S. Chen, and A. Mohaisen, "Delving into Internet DDoS attacks by botnets: Characterization and analysis," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2843–2855, Dec. 2018.

[14] C. Rossow, "Amplification hell: Revisiting network protocols for DDoS abuse," in *Proc. Netw. Distrib. System Security Symp. (NDSS)*, San Diego, CA, USA, Feb., 2014, pp. 1–15.

[15] L. Cai, Y. Feng, J. Kawamoto, and K. Sakurai, "A behavior-based method for detecting DNS amplification attacks," in *Proc. 10th Int. Conf. Innovative Mobile Internet Services Ubiquitous Comput. (IMIS)*, Fukuoka, Japan, Jul. 2016, pp. 608–613.

[16] M. E. Ahmed, H. Kim, and M. Park, "Mitigating DNS query-based DDoS attacks with machine learning on software-defined networking," in *Proc. IEEE Military Commun. Conf. (MILCOM)*, Baltimore, MD, USA, 2017, pp. 11–16.

[17] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *Proc. Int. Conf. Wireless Netw. Mobile Commun. (WINCOM)*, 2016, pp. 258–263.

[18] Y. Zhauniarovich and P. Dodia, "Sorting the garbage: Filtering out DRDoS amplification traffic in ISP networks," in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, Paris, France, Jun. 2019, pp. 142–150.

[19] M. Han, T. N. Canh, S. C. Noh, J. Yi, and M. Park, "DAAD: DNS amplification attack defender in SDN," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2019, pp. 372–374.

[20] K. Özdinçer and H. A. Mantar, "SDN-based detection and mitigation system for DNS amplification attacks," in *Proc. 3rd Int. Symp. Multidiscipl. Stud. Innovative Technol. (ISMSIT)*, Ankara, Turkey, Oct./Nov. 2019, pp. 1–7.

[21] Y. Zhang and Y. Cheng, "An amplification DDoS attack defence mechanism using reinforcement learning," in *Proc. IEEE SmartWorld Ubiquitous Intell. Comput. Adv. Trusted Comput. Scalable Comput. Commun. Cloud Big Data Comput. Internet People Smart City Innov. (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, Leicester, U.K., Aug. 2019, pp. 634–639.
[22] V. Gupta and E. Sharma, "Mitigating DNS amplification attacks using a set of geographically distributed SDN routers," in *Proc. Int. Conf. Adv. Comput., Commun. Inform. (ICACCI)*, Bangalore, India, Sep. 2018, pp. 392–400.
[23] V. F. Garcia, G. de Freitas Gaiardo, L. da Cruz Marcuzzo, R. C. Nunes, and C. R. P. dos Santos, "DeMONS: A DDoS mitigation NFV solution," in *Proc. IEEE 32nd Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, Krakow, Poland, May 2018, pp. 769–776.
[24] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and elastic DDoS defense," in *Proc. 24th USENIX Security Symp. (USENIX Security)*, Washington, DC, USA, Aug. 2015, pp. 817–832.
[25] A. S. Jose and A. Binu, "Automatic detection and rectification of DNS reflection amplification attacks with Hadoop MapReduce and Chukwa," in *Proc. 4th Int. Conf. Adv. Comput. Commun.*, Cochin, India, 2014, pp. 195–198.
[26] A. Lioy *et al.*, "NFV-based network protection: The SHIELD approach," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, 2017, pp. 1–2.

Matteo Repetto received the Ph.D. degree in electronics and informatics from the University of Genoa in 2004, where he was a Postdoctoral Fellow from 2004 to 2009. From 2010 to 2019, he was a Research Associate with CNIT. Since 2019, he has been a Research Scientist with the Institute for Applied Mathematics and Information Technologies, National Research Council of Italy. He has been involved in many different national and European research projects in the networking area funded by the EC, the Italian MIUR and private organizations, both as researchers and principal investigator. He was the Scientific and Technical Coordinator of the projects "AddreSsing Threats for viRtualIzeD services" (ASTRID - call H2020-DS-2016-2017, grant no. 786922) and "A cybersecurity framework to GUArantee Reliability and trust for Digital service chains" (GUARD - call H2020-SU-ICT-2018, grant no. 833456). He has coauthored over 50 scientific publications in international journals and conference proceedings. His research interests include mobility in data networks, virtualization and cloud computing, network functions virtualization and service functions chaining, network, and service security.

Gianmarco Bruno received the Laurea degree in electronics engineering from the University of Genoa, Italy, in 1999. In 2000, he joined Marconi where he worked in the system design, transmission modeling, and network optimization of optical transport systems for worldwide deployments. Since 2007, he has been working with Ericsson and has been involved in various projects, including the development of high speed optical transport systems and multilayer hierarchical software defined networking (SDN) controllers. He has contributed as Ericsson delegate to ITU-T SG15 and IETF CCAMP standardization bodies. He has been involved in joint industry-academia research projects and in the H2020 AddreSsing ThReats for virtualIseD services (ASTRID) Project. He is currently working in the software development of the Ericsson cloud-native solution for Service Management and Orchestration. He has published in scientific journals, such as *Journal of Lightwave Technology* (IEEE) and technical conferences (OFC, ECOC, and ONDM). He is a (co)inventor of 24 U.S. granted patents in the fields of optical communications and SDN. His research interests are in the area of network modeling and automation.

Jalolliddin Yusupov received the M.S. and Ph.D. degrees in computer engineering from the Politecnico di Torino, Italy, in 2016 and 2020, respectively. He is currently the Head of the Academic and Research Department, Turin Polytechnic University in Tashkent (TTPU), Uzbekistan, where he is an Assistant Professor with the Automatic Control and Computer Engineering Department. His primary research interests include formal verification of security policies in automated network orchestration. His other research interests include modeling, cyber–physical systems, and cloud computing systems.

Guerino Lamanna received the bachelor's degree in computer engineering and the master's degree in computer engineering from the Department of Computer, Communications and Systems Science (DIST), currently Department of Electrical, Electronic and Telecommunications Engineering, and Naval Architecture (DITEN), Faculty of Engineering, University of Genoa in 2005 and 2008, respectively. He is currently a Software Engineer with Infocom, that is part of the Scientific Research and Development Services Industry. He took part in different European research projects in collaboration with international partners. From these cooperations derived many research papers of which he has coauthored. He is also involved in participation with many industries. His main area of interest is networking, virtualization architecture technology, and cybersecurity.

Benjamin Ertl received the B.Sc. degree in computer science from the Technical University of Munich, the M.Sc. degree from the University of Copenhagen, and the Doctoral degree (Dr.-Ing.) from the Karlsruhe Institute of Technology. He is currently working as a Senior Software Developer for cloud platform services and support specialist for technical and user support with Agentscape AG (AGE). As an SME, Agentscape develops and markets basic technology for intelligent software agents and operates managed cloud-native solutions for enterprise customers. AGE further participates in several EU projects, responsible for developing core security and privacy micro-services, context-awareness, and knowledge-centered security-and privacy policy management services.

Alessandro Carrega (Member, IEEE) received the B.S. and M.S. degrees in computer engineering and the Ph.D. degree in green networking from the University of Genoa (UniGe), Genoa, Italy, in 2005, 2007, and 2013, respectively. He is a Software and Network Engineer. He is currently a Researcher with UniGe. He is also a Member of the National Inter-University Consortium for Telecommunications (CNIT) in the Genoa Resource Unit (RU), Italy. He took part in the activities of many national and European projects (e.g., H2020 ASTRID, GUARD, ARCADIA, and INPUT, FP7 IP ECONET, PRIN EFFICIENT, FIRB GreenNet, and FI-Core). He is an active reviewer for many different international journals and conferences (IEEE and ACM). He has coauthored several papers in international conference proceedings. In 2011, he was a Visiting Ph.D. Scholar with PSU, Portland, OR, USA, under the supervisor of Prof. S. Singh as an active member of the FINE2 Italy–USA collaboration project. Finally, he is involved in collaboration with many industries, such as TIM, Broadcom, Nokia, Ericsson, and Huawei, and industrial fora, like GeSI. His research is on networking (energy-aware, performance optimization, and virtualization), software routers, NFV and SDN (OpenFlow), container-orchestration system for automating computer application deployment, scaling, and management (Kubernetes) and cloud computing platforms (Openstack). In 2010, he won the best paper award at the 3rd International Workshop on GreenCom 2010 co-located with the IEEE GLOBECOM.