IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 22, 2021, accepted September 18, 2021, date of publication September 24, 2021, date of current version October 6, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3115523

SmartX Multi-Sec: A Visibility-Centric Multi-Tiered Security Framework for Multi-Site Cloud-Native Edge Clusters

JUN-SIK SHIN[1] AND JONGWON KIM[2], (Senior Member, IEEE)
[1]School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology, Gwangju 61005, South Korea
[2]AI Graduate School, Gwangju Institute of Science and Technology, Gwangju 61005, South Korea

Corresponding author: Jongwon Kim (jongwon@nm.gist.ac.kr)

This work was supported in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korean Government [Ministry of Science and ICT (MSIT)], Artificial Intelligence Graduate School Program [Gwanju Institute of Science and Technology (GIST)], under Grant 2019-0-01842, and in part by the Development of Cloud-Native Key Technologies to Collect and Provide Integrated Training Dataset for Autonomous Driving System under Grant 2021-0-01176.

ABSTRACT Recently, to match the emerging demands for multi-site edge clouds, the cloud-based information and communication technology (ICT) infrastructure is rapidly expanding. To protect distributed edge-based cloud assets from networking-based threats by recognizing suspicious traffic, cloud operators should monitor the overall underlying topology to categorize and identify diversified networking packet traffic, flowing through various paths among virtualized and containerized cloud nodes. Perimeter-based networking security, which employs security appliances in fixed locations, cannot address this visibility challenge. As a result, in this paper, we propose the SmartX Multi-tier Security (Multi-Sec) framework, which aims to provide intuitive and systematic visibility for multi-site edge-cloud security. SmartX Multi-Sec abstracts the underlying networking topology among multi-site edge clusters as multiple onion-ring-based tiers of physical, virtualized, and containerized cloud nodes. It also provides collective DevSecOps automation features for monitoring, visualizing, and filtering targeted networking traffic from the respective tiers of the abstracted networking topology. The resulting flow-centric visibility using SmartX Multi-Sec can be featured with extended Berkeley Packet Filter and eXpress Data Path (eBPF/XDP)-leveraged lightweight flow capture and filtering, three-dimensional onion-ring visualization, and automated deployment of DevSecOps functions. By integrating these features, the Proof-of-Concept (PoC)-version of the SmartX Multi-Sec framework is realized to verify the flexible and scalable flow-centric security for multi-site cloud-native edge clouds.

INDEX TERMS Automated function deployment, lightweight flow capture and filtering, multi-site cloud-native edge clouds, security-oriented flow-centric visibility, three-dimensional visualization.

I. INTRODUCTION

With the rapid growth of the Internet of things and 5G mobile networks, edge computing is widely adopted to address demanding resource requirements for AI-leveraged edge services such as high networking bandwidth, low latency, and security improvement [1]–[4]. Along with this technology trend, by intensively adopting virtualization and containerization, cloud-native-style clouds are becoming popular for emerging information and communication technology (ICT) infrastructure [5], [6]. Thus, to maximize resource efficiency and flexibility while satisfying the intense requirements of diversified services, edge clouds are dominantly adopting open-source-based cloud-native computing [7].

Typically cloud-native edge clouds are built over-complicated networking topology because of geographical separation and network isolation (i.e., virtualization) for virtualized and containerized application services. In addition, distributed edge clouds, interconnected with numerous end things and people, are exposed to potentially dangerous external devices via many vulnerable access points. Consequently, the complicated underlying topology naturally leads to diversified edge-cloud networking paths, which can be abused as wide attack surfaces by suspicious behavior flows.

The associate editor coordinating the review of this manuscript and approving it for publication was Yassine Maleh.

In addition to the north-south traffic with hidden external attacks, east-west traffic among internal entities (e.g., physical, virtualized, and containerized nodes) can compromise the multi-site edge clouds from inside.

Detecting suspicious behaviors of malicious flows and estimating their impact coverage could be overwhelmingly difficult in distributed clouds over complicated networking topology. For flow-based detection, the whole networking traffic sent from (or received by) peering networking ports should be captured, collected, and processed by exploiting the limited resources of edge clouds. Then, to estimate the impact coverage and effectively block the attacking threats, the collected packet traffic data and the status of respective peering ports should be intuitively visualized for the DevSecOps operators of multi-site edge clouds.

Without systematic visibility supports, timely reactions against attacking threats cannot be made, and belated reaction incurs resource wastes. Actually, K-ONE Playground trial has been a real-world testbed example for networking-based security threats since 2015 [8], as preliminary multi-site cloud-native edge clouds. Due to the complexity of underlying networking topology, we have wasted time and human resources to identify specific victims and block them from the attack attempts. From this long-term miniaturized experience, the main lesson for the continuous operation of multi-site edge clouds is to maintain the well-designed systematic visibility and apply the organized visibility for flow-centric protection of multi-site edge clouds.

Typically security approaches for edge-cloud clusters are quite diverse since the security for cloud inherently touches the multiple layers of physical, virtualized, and containerized cloud nodes. The resulting security options for protecting edge-cloud infrastructure and services are thus ranging over various concepts, tools, and schemes [9]–[17]. The conventional perimeter-based defense, relying on dedicated security appliances at the infrastructure boundary, does not effectively address attacks originating inside the perimeter. As an alternative, Linux-native flow management features, such as iptables (i.e., netfilter) and traffic control, are widely adopted to protect physical/virtual/container-layer networking interfaces (e.g., ports). These flow management tools typically manage a shared long list of filtering rules for all ports of edge-cloud cluster nodes, which is subject to the burden of effectively sharing the long list. This burden and other limitations reduce its usefulness for multi-site edge clouds, where the management complexity and resource consumption are not trivial due to scalability challenges.

Meanwhile, many novel security schemes for detection accuracy and management effectiveness have been proposed [9]–[13]. Some of the suggested schemes are touching the use of edge as defense belts for a centralized cloud by mitigating DDoS (distributed denial of service) attack or proactive ML (machine learning)-based IDS (intruder detection system). The flexibility of security function chaining, SmartNIC-accelerated security functions, and a hierarchical architecture with security protocols are also introduced. However, leveraging these schemes directly to the multi-site edge clouds for acquiring systematic visibility is not practically affordable due to the wide-range differences in target scenarios and users, underlying networking topology, and cluster node assets to be protected.

In summary, to overcome the topological complication of distributed edge clouds and maintain their continuous operation, it is essential to establish organized and systematic visibility over the targeted cloud infrastructure. Our initial effort, named SmartX MVF (multi-view visibility framework) [18]–[23], has targeted establishing multi-layer visibility over software-defined networking-enabled multi-site cloud-based nodes. SmartX MVF attempts to organize unified multi-layer visibility of the underlay network, physical and virtualized nodes, inter-connecting flows, and application workloads. However, due to the inherent complexity of attempted up to 5 layers of visibility, the achieved multi-layer visibility has been limited to basic-level framework validation emphasizing layered visibility collection and two-dimensional onion-ring visualization. Also, the issue of how to leverage the collected visibility toward the secured operation of multi-site clouds is not yet explored.

Thus, this paper explores an intuitive and systematic solution for a visibility-centric and multi-tiered security framework that can be further customized and scaled for complicated multi-site edge-cloud security. By focusing on the systematic protection for multi-site cloud-native edge clouds, we propose SmartX Multi-tier Security (Multi-Sec) framework as a unique visibility-centric and multi-tiered framework. The main contributions behind the proposed SmartX Multi-Sec can be summarized as follows:

- We list the SmartX Multi-Sec framework requirements for visibility-centric simplification and multi-tiered scalability. The proposed SmartX Multi-Sec framework abstracts the underlying networking topology among multi-site edge clusters as several layers of physical, virtualized, and containerized cloud nodes to meet the requirements. The adopted abstraction simplifies the complicated underlying topology and associated virtualization/containerization of cluster nodes into hierarchical multiple tiers of onion-style rings. It effectively categorizes and identifies diversified networking packet traffic by associating them with the peering networking ports of physical/virtualized/containerized cluster nodes.
- The proposed framework also provides collective DevSecOps automation features that can monitor, visualize, and filter targeted networking traffic from the respective tiers of the abstracted networking topology. The resulting flow-centric visibility employing SmartX Multi-Sec can be featured with lightweight eBPF(extended Berkeley Packet Filter)/XDP(eXpress Data Path)-leveraged flow capture and filtering, three-dimensional (3D) onion-ring visualization, and automated deployment of DevSecOps functions. That is, based on the abstracted visibility visualization,

the framework can flexibly capture and filter networking traffic by associating eBPF/XDP-based tiny-sized DevSecOps functionalities with the targeted peering ports. In addition, the framework provides three-dimensional onion-ring-style visualization that directly depicts the simplified underlying networking topology along with the security status of respective peering ports. With these features, the proposed framework supports the easy and semi-automated repetition of DevSecOps coordination steps, which spans the widely-scoped monitoring, visualization, and filtering of networking traffic.
- By integrating the proposed framework and prototyping the above features, the PoC(Proof-of-Concept)-version of the SmartX Multi-Sec framework is realized to verify the flexible and scalable flow-centric protection for multi-site cloud-native edge clouds. The recognized prototype implementation over K-ONE Playground and its verification results highlight the effectiveness of the proposed framework design and the feasibility of the proposed framework equipped with the respective DevSecOps features for multi-site edge-cloud security.

The rest of this paper is organized as follows. Section II explains our real-world testbed implementation for multi-site cloud-native edge clouds, followed by functional requirements for protecting them. In Section III, to satisfy the given requirements, the overall design of the SmartX Multi-Sec framework and its key components are detailed. Then, the prototype implementation of the proposed framework is explained in Section IV, together with feasibility verification results. In Section V, we discuss the potential use cases of the proposed framework and related work on securing multi-site edge clouds. Finally, we conclude this paper in Section VI.

II. BACKGROUND AND REQUIREMENTS

Edge clouds can be typically constructed in different configurations due to the various resource requirements of target AI-leveraged edge services. Thus, this section describes K-ONE Playground as our miniaturized real-world testbed of multi-site cloud-native edge clouds to assume the underlying networking topology for flow-centric visibility. We list functional requirements to monitor, visualize effectively, and filter networking traffic among distributed physical/virtualized/containerized cloud nodes based on the assumption.

A. K-ONE PLAYGROUND: A MINIATURIZED TESTBED FOR MULTI-SITE CLOUD-NATIVE EDGE-CLOUDS

Motivated by the increasing interest in cloud-native edge clouds, we have operated K-ONE Playground since 2015 to support domestic researchers to realize cloud-based DevSecOps services [8]. K-ONE Playground consists of three infrastructure tiers: a centralized core cloud, distributed edge clouds, and diversified end-things (Fig. 1). The core and edge clouds are built as clusters of physical cloud assets (i.e., physical servers and networking switches). To orchestrate cloud

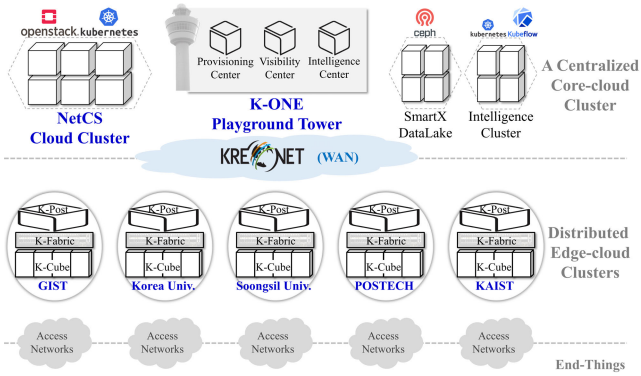

FIGURE 1. The hierarchical networking topology of K-ONE playground.

assets and services effectively, K-ONE Playground employs the systematic structure for operations of SmartX Playgrounds [24], which contains a centralized playground tower and distributed security posts. K-ONE Playground tower at the centralized location manages the entire infrastructure by employing SmartX automation centers that are respectively DevSecOps-based software collections for provisioning, visibility, orchestration, and intelligence tasks. Security posts monitor and control adjacent cloud nodes and services in respective edge clusters by utilizing open-source DevSecOps automation tools. By coordinating the security posts, the playground tower can expand its management coverage to all edge clusters.

The K-ONE Playground was also refined based on the concept of composable playground to effectively support multiple tenants who may demand different networking topologies, software packages, and resource requirements. K-ONE Playground contains feature components such as networking plane separation, a box deployment tool, and a resource-centric visibility tool to provide customized testbeds from the limited resources. K-ONE Playground employing these features allow cloud operators to flexibly deploy and monitor physical/virtualized/containerized cloud nodes over distributed clusters.

B. REQUIREMENTS FOR PROTECTING MULTI-SITE CLOUD-NATIVE EDGE CLOUDS

In addition to geographical separation among edge-cloud clusters, adopting virtualization and containerization, which create many networking ports for virtual overlay networking to interconnect among virtualized/containerized cloud nodes, can incur the networking topology. Monitoring, visualizing, and filtering networking traffic of respective networking ports for protecting cloud assets are overwhelmingly difficult in multi-site cloud-native clusters over the complicated networking topology. Cloud nodes with networking ports that are not properly protected can be easily vulnerable targets. Furthermore, once security threats successfully compromise cloud nodes through the vulnerable ports, the attackers can

exploit the nodes as internal bases for attacking internal cloud assets and services from inside.

Zero-trust security can be a promising conceptual approach to protect the complicated networking of multi-site cloud-native edge clouds from networking-based threats. The zero-trust concept assumes all entities can be suspicious. So DevSecOps operators should monitor, verify, and control all entities all the time. This concept may look straightforward, but its realization can be varied depending on target entities and attacking threats. When it comes to multi-site cloud-native edge clouds, we can consider all physical, virtualized, and containerized cloud nodes as suspicious entities. DevSecOps operators demand scalable and flexible flow-centric visibility that monitor and filter networking packet traffic sent from (received by) cloud nodes. Furthermore, visualization is also significant to timely react against threats because a reaction can begin only after estimating areas affected by the detected threats.

Inspired by the zero-trust concept, we can consider functional requirements of a visibility-centric framework to effectively monitor, visualize, and block networking traffic for multi-site cloud-native edge clouds as follows:

- **R1 (Requirement #1). Providing scalable flow-centric visibility with an abstracted visualization:** A centralized place should collect, store, and process massive amounts of monitoring data for flow-based visibility. The visibility workloads can exploit huge portions of limited computing, storage, and networking resources in multi-site edge clouds. In addition, a security framework should uniquely identify respective peering ports, and collect networking traffic from them. Furthermore, visualizing the security status of peering ports in text formats such as lists and tables cannot help DevSecOps operators intuitively estimate attacking areas by security threats. Drawing the complicated topology in the forms of trees or graphs can easily increase their sizes and complexity. Therefore, flow-centric visibility for multi-site cloud-native edge clouds should employ a systematic abstraction that simplifies the underlying networking topology among virtualized and containerized cloud nodes to cope with the increasing scale. Edge-cloud security should distribute visibility workloads to edge clusters to alleviate the centralized resource consumption. Also, an intuitive visualization for edge-cloud security should depict the abstracted topology and the security status of respective peering ports on the same screen for DevSecOps operators to recognize and understand suspicious behaviors.
- **R2 (Requirement #2). Supporting a flexible and semi-automated deployment of DevSecOps functions:** To monitor and filter packet traffic from cloud nodes, DevSecOps operators need to flexibly deploy software functions that can capture and block networking packets. Because of the topology complication, manual and script-based configuration may waste human resources and be susceptible to human errors. Thus, the visibility-centric framework should support flexible and automated monitoring and filtering functions on the intended peering ports of physical/virtualized/containerized cloud nodes. For this feature, the framework should provide a unified interfacing method for flexible and easy function deployment. In addition, based on the input, the framework should conduct multiple configuration steps for remote cloud nodes, such as installation/configuration of basic software packages and functions, executing functions, and updating option values.
- **R3 (Requirement #3). Utilizing lightweight DevSecOps functions for networking traffic capture and filtering:** DevSecOps functions associating with peering networking ports should directly handle massive packets in real-time. Furthermore, physical cloud nodes can internally contain many virtualized and containerized cloud nodes. Under massive networking traffic, packet capture and filtering from networking ports of these nodes can waste huge compute and storage resources, which results in disturbing legitimate services. Thus, the visibility-centric framework demands tiny-sized software functions that can be flexibly applied to various networking ports whereas consuming small computing resources.

To satisfy these requirements, we propose the SmartX Multi-Sec framework as a flow-centric visibility framework with collective DevSecOps features that can correspond to respective monitoring, visualization, and reaction steps for edge-cloud security. Furthermore, SmartX Multi-Sec suggests a systematic approach for edge-cloud security and supports further research on intelligent security (e.g., AI/DL-based intrusion detection systems, resource-aware security orchestration) that demands flow-centric visibility data and the DevSecOps features. Even though edge-cloud security may have various technical challenges not listed, SmartX Multi-Sec focuses on addressing the functional requirements.

III. DESIGN OF SmartX MULTI-TIER SECURITY FRAMEWORK AND KEY COMPONENTS

In this section, we design the SmartX Multi-Sec framework in a top-down approach. The overall design includes the concept of SmartX Multi-Sec framework based on onion-ring-style topology abstraction, and its framework architecture with collective DevSecOps components. We also design the details of these components to address the requirements described in Section II-B.

A. OVERALL DESIGN OF SmartX MULTI-SEC FRAMEWORK

Fig. 2 depicts the overall concept of SmartX Multi-Sec. SmartX Multi-Sec can abstract the underlying topology among virtualized/containerized cloud nodes as multiple tiers of onion-ring-style rings. Respective networking ports of the nodes can uniquely correspond to the segments on the rings. Based on this abstraction, we can assume that security attacks

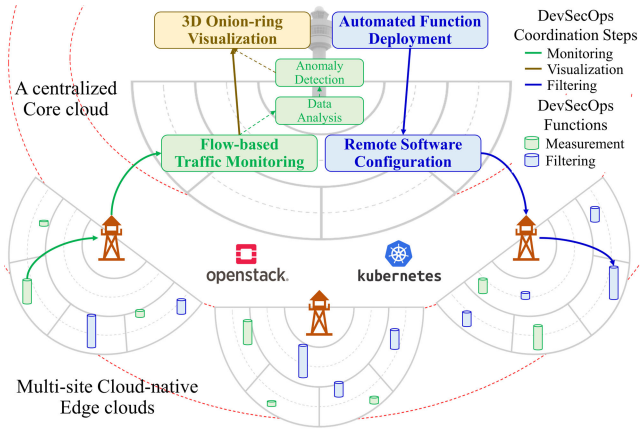

FIGURE 2. The overall concept of SmartX Multi-Sec.

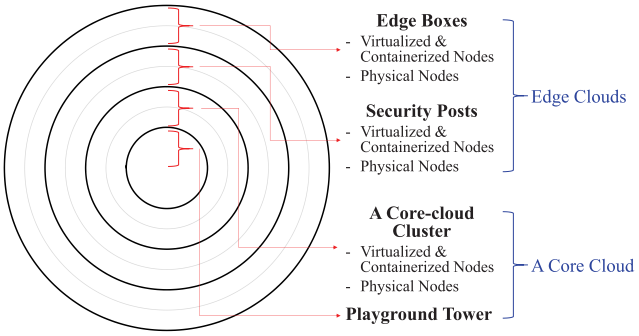

FIGURE 3. An onion-ring-style multi-tiered topology of K-ONE playground.

attempt to find vulnerable networking ports to compromise services and cloud nodes. To detect and react to the attempts, SmartX Multi-Sec can deploy DevSecOps functions, such as measurement and filtering functions, on the respective tiers for flow capture and filtering. Using monitoring data collected from the measurement functions, the visibility center provides 3D onion-ring visualization that can depict both the abstracted topology and security status of respective networking ports on a single graph. The provisioning center can support template-based function deployments to monitor and block targeted networking traffic. SmartX Multi-Sec can allow DevSecOps operators to effectively repeat the security cycle such as monitoring, visualization, and filtering with these features.

When it comes to topology abstraction, we define two types; infrastructure tiers and box tiers. Infrastructure tiers represent a hierarchical relationship of distributed cloud clusters and their operational elements. A playground tower, clusters for core cloud, security posts, and clusters for edge clouds can correspond to the respective tiers in order from the center to the edge. In this relationship, the inner tiers can typically monitor and control the outer tiers. For example, the playground tower can manage all edge clusters, the posts can manage adjacent cloud nodes, and DevSecOps services in a core cloud can coordinate distributed edge services. Thus, inner entities tend to have higher significance than outer entities in DevSecOps operations, so DevSecOps operators may ensure higher-level protection for inner entities by deploying more functions along the networking paths.

Box tiers represent a nested structure of physical, virtualized, and containerized cloud nodes. Notice that SmartX Multi-Sec uses the term *boxes*, defined in SmartX Playgrounds, to refer to cloud nodes. In typical cloud-native-style clouds, virtual and container boxes (e.g., virtualized and containerized cloud nodes) can be nested together depending on operation policies and service requirements. However, handling these indefinite combinations can be impractical. Therefore, in this paper, SmartX Multi-sec only considers networking ports visible in operating systems of physical cloud nodes. Fig. 3 shows the multi-tiered topology of cloud-native edge clouds for SmartX Multi-Sec, which combines the definitions of infrastructure and box tiers.

In addition, we define a dot notation <*cluster*>.<*physical box*>.<*networking port*>.<*function*> to uniquely identify the respective tiers and functions. For example, *gj.k1-gj1-cube1.eno1.measure* can point out a DevSecOps function *measure* deployed on the networking port *eno1* in the physical box *k1-gj1-cube1* in the *gj* cluster. Using this notation, SmartX Multi-Sec can deploy and update DevSecOps functions on exact networking ports.

In the following subsections, we detail the design of essential components of SmartX Multi-Sec framework such as visibility center, provisioning center, and eBPF/XDP-leveraged DevSecOps functions.

B. VISIBILITY CENTER FOR SCALABLE FLOW-CENTRIC MONITORING AND VISUALIZATION

SmartX Multi-Sec framework can employ the visibility center to support flow-centric visibility tasks for multi-site cloud-native edge clouds. For the visibility center, we design a 3D onion-ring graph structure for the visualization as depicted in Fig. 4. The horizontal surface of the graph can show the multi-tiered topology that looks like the cross-section of an onion. Also, the graph can be vertically rotated to show vertical heights of the ring segments that correspond to vulnerability scores of the respective ports.

The visibility center should internally collect, store, analyze, and stage monitoring data to continuously supply data to the visualization. The respective tasks can be implemented as software-based visibility modules. The data collection module can acquire monitoring data captured by measurement functions. The storing module can modify the obtained data to be compatible with a pre-defined schema and store them into databases. The data analysis module mainly focuses on generate security-oriented information such as vulnerability levels of cloud nodes, services, and tenants by clustering, classifying, and identifying networking traffic flows. The staging module can prepare the visibility data for other modules (e.g., the visualization module for 3D onion-ring-based graph) to instantly utilize.

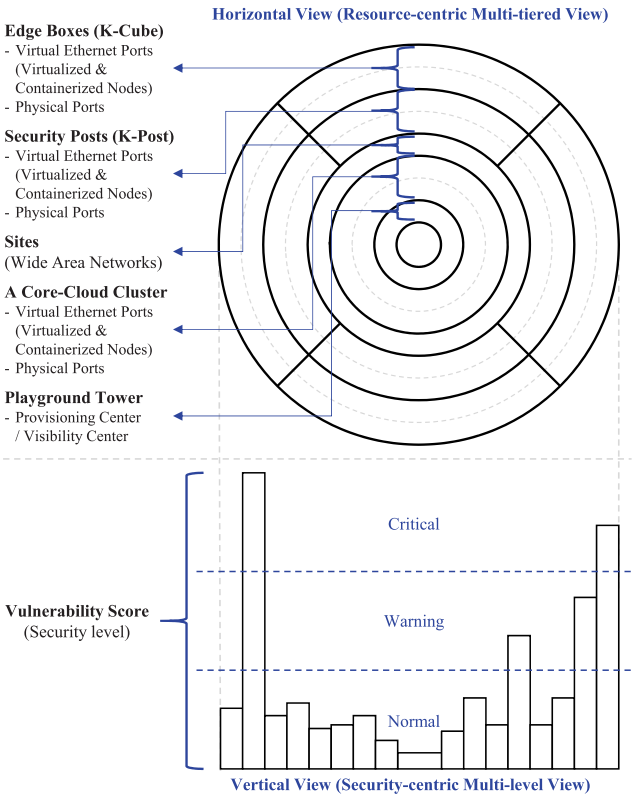

FIGURE 4. Design of 3D onion-ring visualization.

Header fields of raw packets cannot directly represent time-related networking patterns such as the duration of flows, packets/bytes per second, and inter-packet arrival time. Thus, typical intrusion detection systems (IDS) employ flow-based monitoring that obtains flow features from captured raw packets to detect suspicious patterns based on signature and policies. Likewise, public IDS datasets such as UNSW NB-15 [25], CICIDS-2017 [26] provide collections of flow features rather than raw packet data. For flow-based monitoring, SmartX Multi-Sec should manage flow caches to maintain flow expiration and generate additional features.

By employing these modules, the visibility center can continuously supply the collected data into the visualization module. Then, the 3D onion-ring graph can highlight impacting areas affected by suspicious traffic over the multi-tiered topology. Consequently, operators can easily recognize security threats and estimate the scopes for further inspection and reaction.

C. PROVISIONING CENTER FOR FLEXIBLE AND AUTOMATED FUNCTION DEPLOYMENT

To design function deployment for SmartX Multi-Sec, we should define where eBPF/XDP-leveraged DevSecOps functions can be associated for flow capture and filtering. Linux kernel has several candidates: utilities (e.g., Netfilter and traffic control) and kernel functions (e.g., *ip_rcv()* and *ip_send()*). These Linux-native features can handle all networking traffic visible in the kernel, regardless of packets' sources and destinations. However, these features manage security rules of networking ports in a lengthy list (or several lists). Thus, they can be susceptible to a single-point-of-failure and the management complexity of security rules. For that reason, they can reveal limitations in scalability, resource efficiency, and processing performance. Therefore, SmartX Multi-Sec can directly associate DevSecOps functions with Ethernet interfaces of cloud nodes. Filtering rules directly bind to respective ports decreasing the average number of inspection rules for filtering packets.

However, the increasing number of DevSecOps functions in multi-site cloud-native edge clouds can increase the complexity in terms of deployments. Manual and script-based deployment should manually configure and verify each step such as access to remote cloud nodes, finding exact peering ports, installing DevSecOps functions with dependent packages, and modifying the detailed configuration. This approach can be error-prone and waste a huge amount of human resources.

To address the complication, the provisioning center of SmartX Multi-Sec can support template-based function deployment. As an input to describe the desired function topology, a provisioning template for SmartX Multi-Sec should contain three attributes of DevSecOps functions: identifiers of networking ports, function types, and types of deployment tasks. Firstly, the identifiers follow the dot notation defined in Section III-A, which helps the provisioning center identify networking peering ports. Next, we can choose between measurement (i.e., capture) and filtering as a function type, as both functions require different setups. Finally, there are three types of deployment tasks: *compose*, *update*, and *release*. For the *compose* task, the provisioning center newly deploys functions on specified ports. The *update* task can change options and filtering rules of functions. The *release* task removes functions working on specified ports. By listing functions with these attributes, a provisioning template can depict a desired topology of functions.

SmartX Multi-Sec can automatically conduct a series of provisioning (i.e., installation and configuration) steps based on the template. To effectively implement the automated deployment feature for the provisioning center, we can define provisioning steps such as template interpretation, DevSecOps tool selection, DevSecOps tool execution, and installation/configuration. In the template interpretation step, the provisioning center can understand the desired result of deployment by reading a given template. In the tool selection step, the provisioning center can select DevSecOps automation tools among available candidates. The provisioning center can call the interfaces of the selected tools and pass parameters to execute configuration tasks. Then, the invoked tools conduct the configuration step by remotely installing and configuring software such as kernels, libraries, packages, and function source codes. DevSecOps operators only need to describe functions in a provisioning template for flexible

monitoring and filtering on targeted networking ports thanks to the template-based function deployment.

D. LIGHTWEIGHT DevSecOps FUNCTIONS FOR FLOW CAPTURE AND FILTERING

After deployment by the provisioning center, SmartX Multi-Sec DevSecOps functions associated with peering ports can capture or filter networking packet traffic. To implement tiny-sized DevSecOps functions for flow capture and filtering, we can leverage Linux eBPF and XDP. Linux eBPF provides a set of libraries that allows dynamic injection of codes from userspace into various kernel events (i.e., hooks) such as kernel functions, system calls, tracepoints, and sockets [27], [28]. Meanwhile, XDP provides special hooks and libraries for eBPF kernel programs to rapidly forward, drop, and redirect packets received at networking ports [29]. XDP hooks include the very early point in the kernel networking stack, device drivers, and SmartNIC. SmartNIC is a short expression of a smart networking interface card that can offload networking-related workload, such as packet en(de)capsulation, monitoring/filtering, shaping, and routing, from CPU to its networking processors. Also, eBPF functions associated with XDP hooks can be executed in one of XDP modes such as generic, native, and hardware offload modes, depending on the type of peering ports. In XDP generic mode, the kernel programs can be freely associated with standalone Linux networking devices such as physical networking ports and virtual Ethernet (i.e., veth) interfaces, but processing performances are significantly lower than other modes. The kernel programs in XDP generic mode can be only associated with device drivers of physical NIC, thus they inspect before packets enter the kernel networking stack. And XDP offloaded mode supports the kernel programs that can directly handle raw packets inside the receiving queue in SmartNIC.

An eBPF program for kernel older than 5.2 can contain at most 4096 instructions without loops. The restriction can ensure the DevSecOps functions quickly to finish packet processing within a finite period, consuming small computing resources. SmartX Multi-Sec can deploy eBPF/XDP-leveraged DevSecOps functions for both flow capture and filtering in the same way. Also, thanks to veth interfaces in the Linux kernel, a single eBPF/XDP-based function can be flexibly utilized for physical, virtual, and container boxes. Therefore, SmartX Multi-Sec employing eBPF/XDP-leveraged lightweight DevSecOps functions can satisfy the flexibility to handle different networking ports in typical cloud-native-style clouds.

Fig. 5 depicts the basic design of DevSecOps functions. Both measurement and filtering functions can be divided into userspace codes and kernel codes. In addition to kernel code injection, userspace programs can interconnect kernel programs with external entities. The userspace programs can get kernel-captured data and send it to the visibility center for monitoring purposes. Inversely, the userspace programs for the filtering functions can update filtering rules received from the provisioning center into the kernel programs.

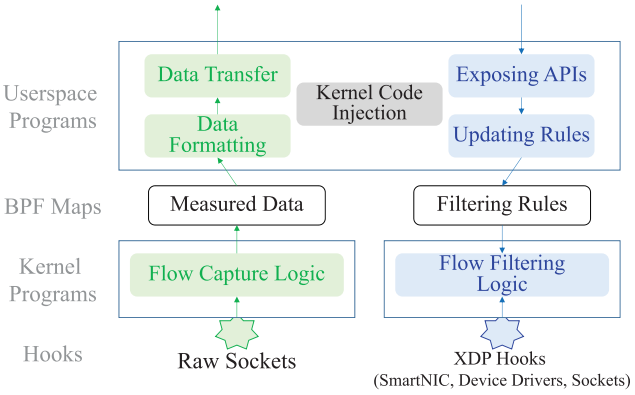

FIGURE 5. Design of SmartX Multi-Sec DevSecOps functions for networking flow capture and filtering.

When it comes to kernel programs, working procedures of the measurement and filtering functions can be straightforward. These functions can be executed when a packet hits the associated networking ports. The kernel programs of the measurement functions can take pre-defined header fields from the packet and send the data to the userspace programs. Inversely, the kernel programs of filtering functions can look up filtering rules written by userspace programs, match the header fields against the rules, and drop the matched packet. These functions can employ eBPF maps that are data structures shared between eBPF-based userspace and kernel programs for communications.

Even though eBPF-based functions are lightweight, capturing respective packets can consume huge computing and storage resources and even incur significant networking delays. Among various factors, a considerable amount of raw packet data transferred from kernel to userspace can be a significant factor that incurs the performance limitation. For that reason, SmartX Multi-Sec can directly generate flow features from raw packets in the Linux kernel using eBPF maps as kernel-level flow caches. In the perspective of the userspace program, huge traffic can be summarized as a few entries of flow features. Therefore, flow-based capture can reduce the amount of monitoring data. And the reduction can alleviate the performance limitations and resource wastes, resulting in improving the scalability.

IV. PROOF-OF-CONCEPT IMPLEMENTATION AND FEASIBILITY VERIFICATION

This section implements a PoC prototype of the SmartX Multi-Sec framework for the K-ONE Playground. Based on a scenario, we describe how the respective components of the SmartX Multi-Sec framework can address the functional requirements for the feasibility verification.

Fig. 6 depicts an example scenario. The dots, drawing on the multi-tiered underlying topology of the K-ONE Playground, correspond to the networking ports of cloud nodes. For simulation, we simultaneously generate ICMP flood DoS attack traffic traversing different networking paths. Flood

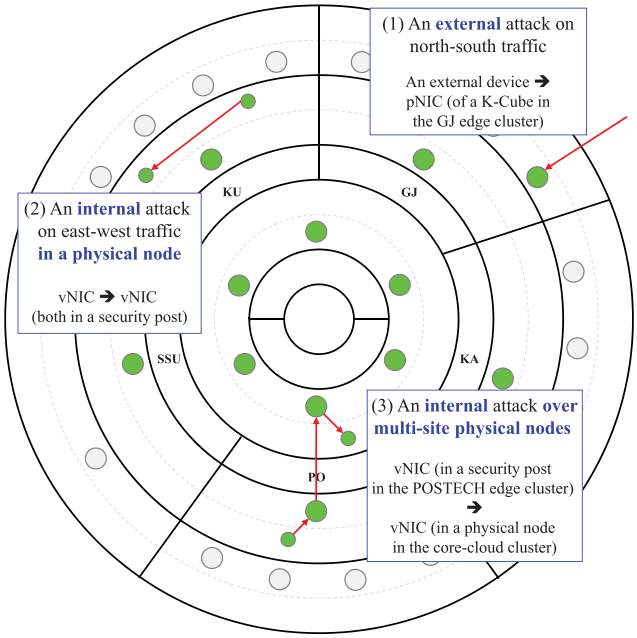

FIGURE 6. An example attack scenario to verify the feasibility of SmartX Multi-Sec. (GJ: GIST, KU: Korea university, SSU: Soongsil university, PO: POSTECH, KA: KAIST).

DoS attacks are very challenging for monitoring and filtering networking-based threats. Massive attacking traffic can easily exhaust computing, storage, and networking resources for a visibility-centric security framework to capture, process, and store respective packets. The red arrows represent the paths of the respective attacks. The first path at the top corresponds to availability attacks from external dangerous devices. The second path represents internal attacks between virtual boxes in a physical box. And the third one at the bottom corresponds to attacks traversing multiple tiers of distributed clusters.

When it comes to testbed configuration, we utilize the minimal number of boxes (i.e., cloud nodes) to clearly show how respective DevSecOps features of the SmartX Multi-Sec framework work. In addition to the boxes involving in the attacks, the provisioning/visibility centers, the core-cloud cluster, and security posts should always be in working status for continuous operations. We utilize *hping3* to send huge ICMP packet traffic having 65000 bytes of payload, which forces victims to waste more computing resources for reassembling fragmented packets. A physical CPU core can approximately generate 2-3 Gbps ICMP traffic using *hping3* at most, so we adjusted the traffic size of attacks with the number of cores and nodes.

Based on the scenario, we verify the respective features to show the feasibility of SmartX Multi-Sec with its flexibility and scalability as a basic visibility-centric framework for edge-cloud security.

A. THE FLOW-CENTRIC MULTI-TIERED VISIBILITY FRAMEWORK WITH 3D ONION-RING VISUALIZATION

Based on the design of the visibility center, we implement visibility modules such as data collection, storing, analysis, staging, and visualization modules. Notice that SmartX Multi-Sec employs the software structure of SmartX MVF with its leveraged open-source software for effective implementation of flow-centric visibility.

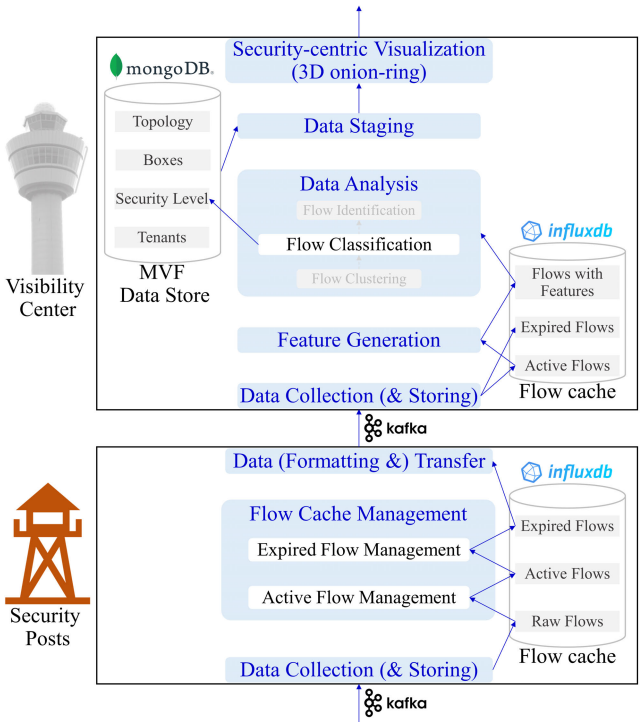

FIGURE 7. The working procedure of the visibility center.

Fig. 7 depicts the structure of visibility modules in detail. The arrows represent a data path where visibility data are transferred, stored, and processed. The visibility center leverages Apache Kafka for message queues, MongoDB for an MVF data store, and InfluxDB for a flow cache. The MVF data store contains management information such as physical infrastructure topology, login credentials, and a tenant list. Also, it stores resource-centric visibility data such as lists of physical, virtual, and container boxes with their resource status. The data collection module takes flow data from message queues, where measurement functions transfer the captured data continuously. The data storing module modifies the format of the collected data and stores them in the flow cache. Notice that DevSecOps functions identify respective networking flows in a physical node with a 5-tuple identifier that consists of source/destination IP addresses, source/destination ports, and protocol. However, the 5-tuple identifier cannot be used to uniquely identify flows in multi-site cloud-native edge clusters since virtual overlay networks in different physical nodes and clusters can have duplicated private IP addresses. Thus, the visibility center and the security posts use the 6-tuple identifier that contains the additional *where* field written in the dot notation.

The flow cache management module manages active and expired flows as separated tables in the flow cache for flow-centric visibility. The module combines the collected flow

features into entries matching the 6-tuple in the active flow table. The module also expires active flows by moving the entries into the expired flow table. In this implementation, expired flows are idle for 60 s from the last received packet or include a connection termination message (i.e., TCP FIN packet). The feature generation module then develops 9 time-related features of the active flows, such as flow duration, packet per second, and byte per second. The visibility center finally acquires the 6-tuple identifier and 22 features of each uni-directional flow used for intrusion detection.

Due to the modular design of SmartX Multi-Sec, security posts accommodate the collection module and flow management module to distribute the centralized visibility workloads, as depicted in Fig. 7. Security posts pre-process flow features collected from adjacent cloud nodes so that the visibility center can alleviate resource consumption for flow management and data transfer. The data analysis module calculates the security levels of networking ports using the processed flow features. In this PoC, we employ a simple method for the data analysis module to quantify vulnerability scores of the flood DoS attack flows based on flow bytes per second: 10 Gbps equals 100 and 0 bps equals zero, and a more significant score means the flow is more suspicious. The analysis module stores the quantified results in the MVF data store.

The data staging module combines different monitoring data stored in the MVF data store into a single data object for the visualization module. After finishing the data analysis, the MVF data store contains the underlying topology among physical cloud nodes, a list of virtual/container boxes, their resource status, and their vulnerability scores. The staging module firstly adds the list of virtual/container boxes into the underlying topology. It then updates resource status (i.e., normal, failed, shutdown), owners (tenants), and vulnerability scores of the respective boxes. Consequently, the staged object is the underlying networking topology among physical, virtual, and container boxes embedded with vulnerability scores and other information.

The visualization module draws a 3D onion-ring graph by using the staged object. We leverage three.js, an open-source graphic library specialized for 3D visualization, to implement the data visualization module. The module finally illustrates a 3D onion-ring graph on a web-based dashboard, as depicted in Fig. 8. The graph intuitively visualizes the topology of the K-ONE Playground on the onion-ring surface. The gray-colored ring segments correspond to idle boxes that are not used for the example scenario. On the other hand, ring segments for active boxes have different colors and vertical heights, based on their vulnerability scores calculated by the data analysis module.

We generate flood DoS attack traffic based on the scenario and show the visualization changes for the feasibility verification. Fig. 9 shows the 3D onion-ring graph under the traffic. All networking ports on the attack paths are highlighted with their colors and heights. The vulnerability score of the red-colored segment is around 90 (i.e., 9 Gbps),

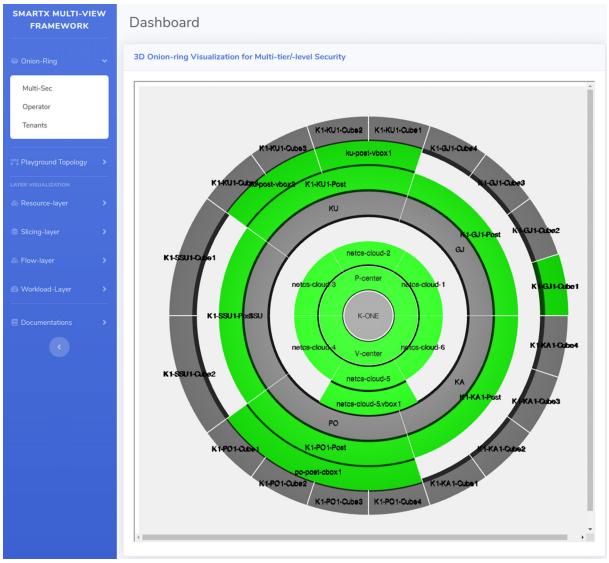

FIGURE 8. A web dashboard of SmartX Multi-Sec for 3D onion-ring visualization.

the orange-colored segments are around 70 (i.e., 7 Gbps), and the yellow-colored segments are around 50 (i.e., 5 Gbps). The visibility center of SmartX Multi-Sec intuitively visualizes the networking ports affected by suspicious networking flows. Thus, we can estimate the impacting areas where we should carefully check to detect suspicious traffic. Once operators react to the threat by deploying filtering functions, the vertical height of the corresponding segments also automatically decreases.

To compare packet-based monitoring with the proposed scheme in terms of scalability, we consider our previous version of an eBPF function that selectively captures 7 header fields and transfers monitoring data in CSV format [20]. For a networking port receiving a uni-directional 10 Gbps flow in 1500-byte MTU, this function should copy 83000 records from kernel to userspace every second. Then, the userspace program should rewrite the records to 80 bytes of a message in CSV string format with additional metadata (e.g., timestamps and node names). Without packet sampling, a uni-directional 10 Gbps flow generates 6.6 MB monitoring data every second. On the contrary, the proposed flow monitoring transfers a 600-byte JSON message for 17 features of each uni-directional 10 Gbps flow.

Employing flow-centric visibility can significantly reduce the usage of WAN resources for edge-cloud monitoring. In addition, the elapsed time from the beginning of an attack to a change on 3D visualization takes 4.81 s in this PoC implementation. The elapsed time and the monitoring traffic reduction show the scalable and continuous monitoring capability of SmartX Multi-Sec.

B. DISTRIBUTED SECURE PROVISIONING TOOL FOR TEMPLATE-BASED DevSecOps FUNCTION DEPLOYMENT

To react against suspicious activities highlighted by the visibility center, DevSecOps operators utilize the provisioning

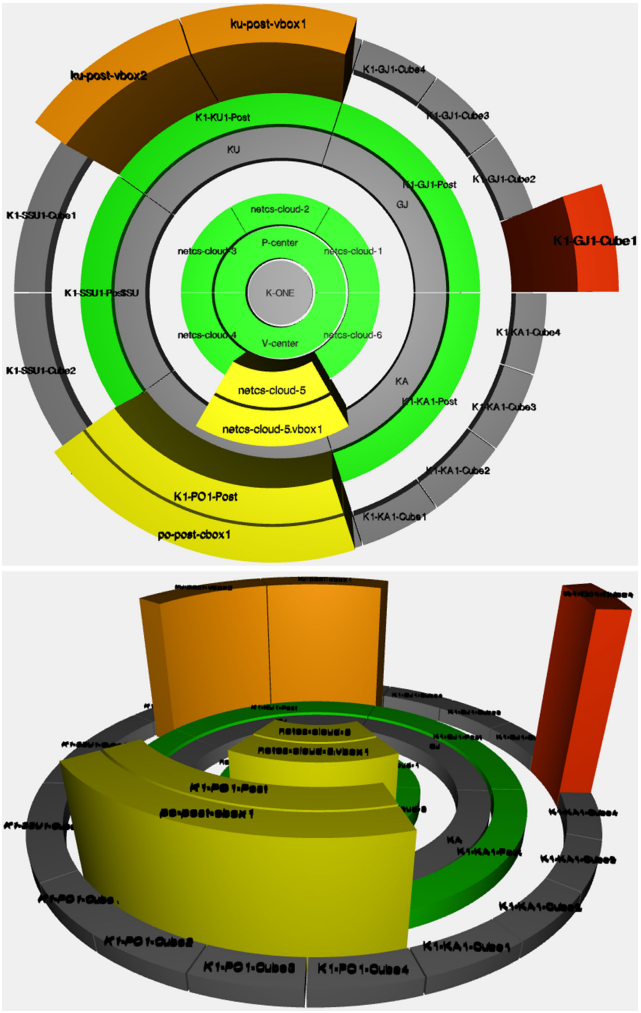

FIGURE 9. The implemented result of security-centric 3D onion-ring visualization for the example scenario.

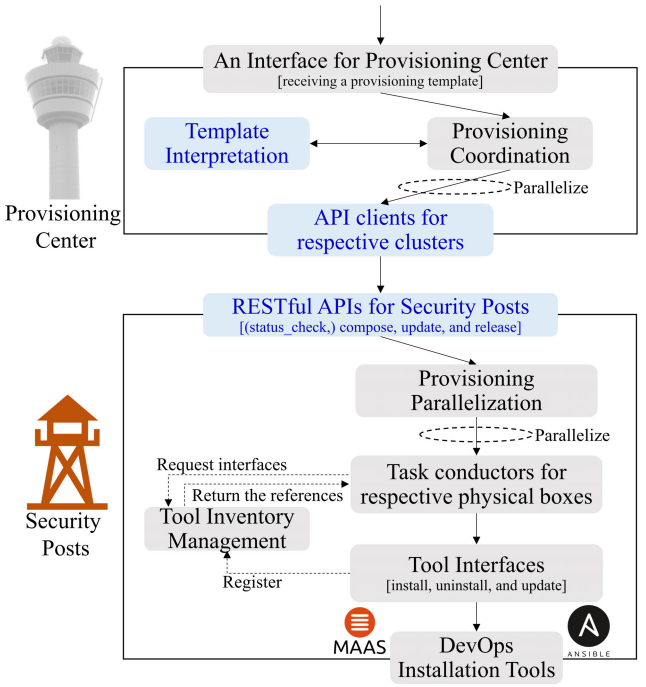

FIGURE 10. The working procedure of template-based function deployment.

```
- tenant: <tenant name>
  task: [ compose | update | release ]

  boxes:
    - name: <box name>
      where: <cluster.box>
      type: [ physical.box | virtual.box ]
      software:
        - name: <software name>
          installer: <software installer name>
          option:
            - <detailed options>

  functions:
    - name: <function name>
      where: <cluster.box.networking_port>
      type: [ multi-sec.measure | multi-sec.filter ]
      option:
        mode: [ generic | native | offload ]
        rule:
          - <filtering rules>
```

FIGURE 11. The basic format of the provisioning template.

center of SmartX Multi-Sec to deploy measurement and filtering functions. For template-based automation, the provisioning center employs Distributed Secure Provisioning tool [30] with additional implementation for deploying SmartX Multi-Sec DevSecOps functions. Fig. 10 depicts the software structure of the provisioning center with a working procedure. The provisioning center covers high-level coordination and then distributed security posts conduct the actual configuration using DevSecOps automation tools. In multi-site edge clouds, respective clusters are typically configured with different L2 (layer 2) networks. The separation of the provisioning capability allows SmartX Multi-Sec to support some DevSecOps automation tools requiring broadcast-based L2 networking, such as DHCP (dynamic host configuration protocol) and PXE (preboot execution environment), with physical boxes. The tool separation also has the advantage of workload distribution and reduction of operation traffic in wide area networks.

The provisioning center receives a template, written in the YAML-based format depicted in Fig. 11, from DevSecOps operators to request function deployment. Function description contains four fields which are *name*, *where*, *type*, and *option* fields. The *where* field specifies peering ports where the functions are deployed in the dot notation, and the *name* field identifies functions on the same port. SmartX Multi-Sec assigns a unique identifier to each function by appending the *name* field to the *where* field (i.e., *cluster.box.networking_port.function_name*). Also, the *type* field defines function types, and the *option* field brings additional parameters for the functions such as filtering rules. A template describes a desired topology of functions by listing the description of functions.

The template interpretation module of the provisioning center reads the desired topology from the template. It adds access information of the physical boxes (i.e., IP addresses and SSH credentials) where these functions will be deployed. Then, the provisioning center sends deployment requests through RESTful APIs of distributed security posts. When receiving requests, security posts create multiple processes as much as peering ports listed in the request. The processes select appropriate API clients for DevSecOps automation tools. We also implement playbooks for Redhat Ansible, a popular open-source DevSecOps automation tool for remote software configuration, to automate configuration steps for function deployment. Selecting and calling the API client for Ansible executes Ansible to automate the function configuration steps based on the playbooks. The configuration steps are accessing target cloud nodes through SSH, installing software packages, copying the source codes of SmartX Multi-Sec DevSecOps functions, and associating the functions with peering ports.

In typical cloud-native operations, cloud operators can access physical cloud nodes but cannot access the inside of virtualized/containerized nodes owned by users. Thus, in terms of function deployment, associating the DevSecOps functions with Ethernet ports of physical cloud nodes is very straightforward. When creating virtualized node, the host operating system (OS) in a physical node creates virtual Ethernet (i.e., veth) interfaces that bridge external networks with the networking ports inside virtualized nodes. Likewise, host OS creates veth interfaces that peer with internal networking ports of containerized nodes. In host OS, both physical ports and veth interfaces are considered as standalone networking devices with which eBPF-leveraged DevSecOps functions can be associated in the same way. Furthermore, all networking traffic going into/coming from the virtualized/containerized nodes must go through the veth interfaces. Thanks to the veth interfaces, the provisioning center employing template-based DevSecOps function deployment supports the flexibility for cloud operators to monitor and filter networking traffic of physical, virtualized, and containerized cloud nodes.

We show how SmartX Multi-Sec easily and flexibly deploy DevSecOps functions to block the DoS attack traffic regarding feasibility verification. By the visibility center, the ring segments of victim boxes were highlighted with heights and colors. To block the traffic, we describe the topology of three filtering functions into a template, as shown in Fig. 12. The *option* field contains initial matching rules and XDP modes. Notice that *gj.k1-gj1-cube1.eno7* is a physical networking port of SmartNIC. The filtering function for this port is executed in hardware offloaded mode, so its filtering performance is highly accelerated without using CPU resources. The rules in the *option* field are injected into eBPF maps for the filtering functions. Using this template as input, the provisioning center automates the configuration of filtering functions to the exact peering ports. Regardless of locations and types of peering ports, deploying functions was

```
- tenant: admin
  task: compose
  functions:
    - name: filter_kpv1vnet1
      where: ku.k1-ku1-post.
             ku-post-vbox1.vnet1
      type: multi-sec.filter
      option:
        mode: generic
        rule:
          - src_ipaddr: 172.30.92.101
            dest_ipaddr: 172.30.92.102

    - name: filter_cnc5eno3
      where: core-cloud.
             netcs-cloud-5.eno3
      type: multi-sec.filter
      option:
        mode: native
        rule:
          - src_ipaddr: 172.30.90.201
            dest_ipaddr: 172.30.90.202

    - name: filter_gc1eno7
      where: gj.k1-gj1-cube1.eno7
      type: multi-sec.filter
      option:
        mode: offload
        rule:
          - dest_ipaddr: 172.30.91.3
```

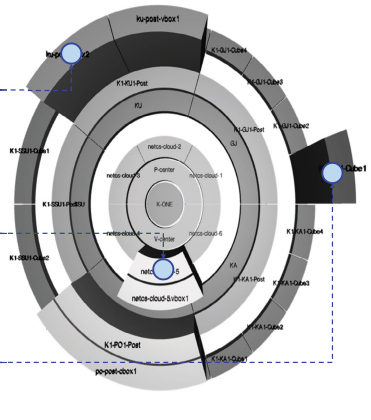

FIGURE 12. A provisioning template to filter the attack traffic in the example scenario.

finished within 30 s. It is worth mentioning that the provisioning center for K-ONE Playground can support the automated deployment of physical/virtual boxes, so we can also remove the boxes causing internal attacks by writing a provisioning template. Consequently, SmartX Multi-Sec employing template-based function deployment supports easy and flexible flow capture and filtering.

C. LINUX eBPF/XDP-LEVERAGED LIGHTWEIGHT MEASUREMENT AND FILTERING FUNCTIONS

To implement SmartX DevSecOps functions for flow capture and filtering, we leverage BPF compiler collection that provides Python libraries and utilities for developers to effectively implement eBPF-based programs for monitoring, networking, and security.

When deploying a measurement function to a networking port, its eBPF kernel program is associated with the kernel function (i.e., *sock_queue_rcv_skb()* kernel function) for the networking port. Thus, the kernel program is executed to handle raw packets when the packets enter the kernel function. The program firstly records the arrival timestamp of the packets in nanoseconds. It selectively takes header fields from layer 2 to layer 4 headers: destination/source IP addresses, destination/source ports, protocol, TCP flags, header length, packet length, etc. The software takes an entry of flow features with the same key from an eBPF map and changes the matching entry using a 5-tuple (i.e., destination/source IP addresses, destination/source ports, protocol) as a key. If the flow is new, then the program stores the new entry to the map. By repeating these steps for every packet, the kernel program generates the 5-tuple identifier and 13 features (e.g., packet sizes, inter-packet arrival time, TCP flags) and 4 additional data (e.g., timestamps) for each uni-directional flow. Meanwhile, the userspace program of the measurement function operates asynchronously after associating the kernel program with the target peering port. The userspace program repeatedly takes the flow features from the eBPF map every second. Then, the userspace program packs the flow features into a

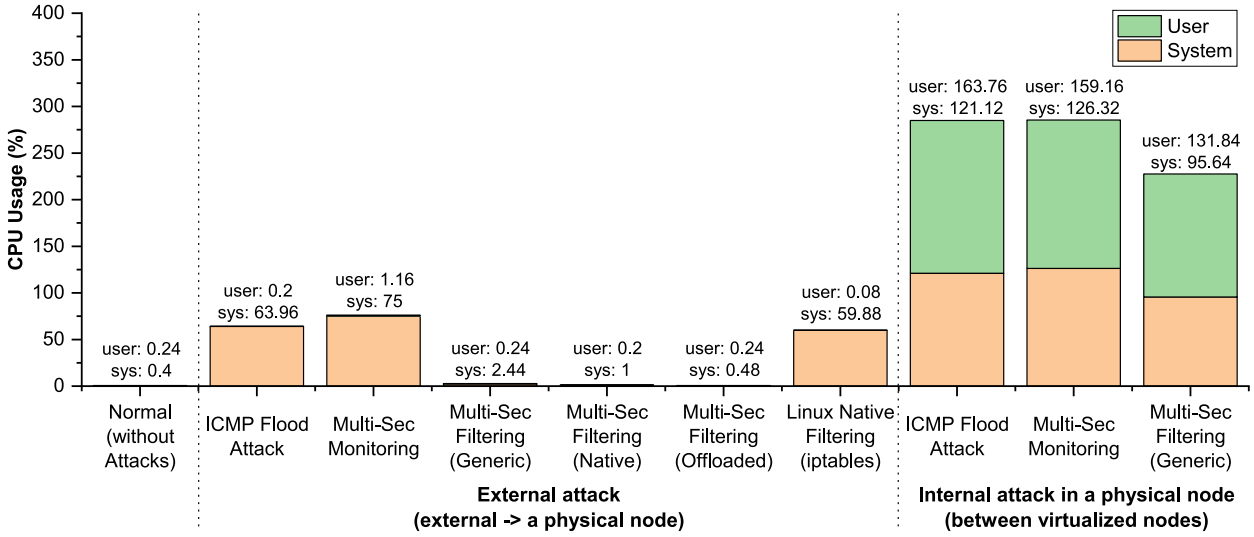

FIGURE 13. CPU utilization of DevSecOps functions under external and internal ICMP flood DoS attacks.

message and transfers it to a message queue of an adjacent security post.

Regarding filtering functions, the userspace program injects kernel programs with one of the XDP modes. Then, the userspace program receives rule update messages from the provisioning center through its RESTful APIs. The rules are converted into C struct (i.e., structure for C language) compatible with eBPF maps for filtering rule lists. Each filtering rule contains 5-tuple of traffic flows. An empty element in the tuple is considered as a wildcard value. The kernel program is invoked when the associated port receives a packet, firstly takes 5-tuple of the packet and compares it with the rule lists. if a matching rule exists, the packet is instantly dropped by the kernel program, and if not, the packet is passed to the normal kernel networking path.

The flexibility of the DevSecOps functions enables fine-grained protection for diversified networking paths in multi-site cloud-native edge clouds. Also, the measurement and filtering functions directly handle packets in the Linux kernel without copying to userspace, which enables rapid packet processing in a resource-efficient way. To verify these advantages, the function deployment by the provisioning center showed the flexibility of measurement and filtering functions in Section IV-B. Also, the performance and resource efficiency of using eBPF/XDP-leveraged packet capture and filtering have been evaluated and discussed in other research works. For example, [14], [31] evaluated the overwhelming performance of Linux eBPF/XDP-leveraged packet capture and filtering compared to *iptables*.

In addition, we evaluate CPU usages of the DevSecOps functions under external attack and internal attack to verify scalability. We utilize the K-Cube box, a physical edge node with 4 CPU cores of Intel Xeon-D processor in K-ONE Playground for both attack cases. In addition, for the internal attack, we create two virtualized nodes having 4 virtual CPUs respectively. Fig. 13 depicts the experiment results. The Y-axis means the average usage percent of four physical cores during each experiment case. For example, 200% means two cores are fully utilized on average.

In the external attack, the physical victim node receives 10 Gbps attack traffic that occupies 97% utilization of a physical port. This attack basically increases 63.5% CPU usage. Associating a monitoring function with the physical port additionally consumes 12% of a core. This result shows that the CPU overhead for monitoring functions can be reasonable under DoS attacks. We deploy filtering functions in different XDP modes. The CPU usages are significantly reduced because the filtering functions drop the attacking packets at the early point of the kernel networking path before assembling the fragmented ICMP packets. Furthermore, filtering functions utilize very small computing resources compared to the normal status. Also, SmartX Multi-Sec employing eBPF-based DevSecOps functions can satisfy the higher scalability than the Linux-native filtering scheme (i.e., *iptables*).

The attacker node continuously sends 2.4 Gbps DoS traffic to the victim node when it comes to the internal attack. Notice that resource usages for networking between virtualized nodes can fluctuate over time. This experiment shows the sudden decrease in 20% CPU usages and 400 Mbps networking bandwidth at most, so we measure the average results of 20 successful repetitions that show consistent CPU usage without networking bandwidth decrease for 60 s. Unlike the external case, we associate DevSecOps functions with the virtual Ethernet interface of the attacker node, not the victim node. An internal attack can quickly drain a physical edge node's limited processing capabilities, as the attack consumes around 285% of CPU consumption. The addition of a monitoring function boosts CPU utilization by about 1%. A filtering function also decreases the CPU usage

of a physical node since the victim node does not utilize the CPU to receive and reassemble the attacking traffic of fragmented packets.

In summary, the CPU usages of various use cases showed that deploying multiple monitoring and filtering functions in a physical cloud node can be feasible. This result guarantees the feasibility as well as scalability of SmartX Multi-Sec employing eBPF-based fine-grained monitoring and filtering.

V. DISCUSSION AND RELATED WORK

In this section, we discuss use cases where SmartX Multi-Sec can aid in protecting multi-site cloud-native edge clouds. We describe related work that proposes similar approaches for edge-cloud security. And we clarify the differences with SmartX Multi-Sec that focuses on flow-centric visibility for protecting multi-site cloud-native edge clusters.

A. DISCUSSION ON USE CASES

The featured advantages of SmartX Multi-Sec can be flow-centric visibility based on the simplified underlying topology of multi-site cloud-native edge clouds, and fine-grained flow capture and filtering by deploying lightweight DevSecOps functions. Based on the advantages, SmartX Multi-Sec can be utilized as a complementing feature to implement several use cases such as micro-segmentation and multi-perimeter defense.

Inspired by the defense-in-depth (DoD) strategy [32], the multi-perimeter defense maintains multiple defense belts over multi-site edge clouds by deploying micro security functions. Multi-perimeter defense focuses on adjusting the accessible coverage of networking traffic and protection levels of cloud assets, based on traffic sources and the importance of destination cloud assets. These perimeters are logically built with DevSecOps functions on the same tiers of multi-site edge clouds. We can apply different policies on the respective perimeters based on the importance of cloud assets and typical attack patterns of the tiers. For example, the red-dashed lines in Fig. 2 can be infrastructure-scale multi-perimeters. We can configure the bottom perimeter, built over physical SmartNICs of cloud nodes, to handle flood DoS attacks from external devices. Among cloud nodes in edge clusters, the midst perimeter can mainly block networking traffic of detected attacks based on blacklists. The top perimeter can restrict tenants and services not registered in whitelists from accessing core clouds and the playground tower for the most important assets. The multi-perimeter defense can be the main application for K-ONE Playground, where gives tenants enough freedom to realize their services over edge clouds whereas highly protecting core clouds and the playground tower.

SmartX Multi-Sec can also be utilized to realize micro-segmentation. Micro-segmentation separates cloud nodes into logical segments based on tenants and services and applies different security policies to the segments by coordinating security functions. Thanks to the fine-grained networking monitoring and filtering, SmartX Multi-Sec can create logical segments by using different security policies to respective sets of security functions. Micro-segmentation and the multi-perimeter defense are conceptually similar in terms of separating edge clouds and applying different policies. Multi-perimeter defense basically assumes the multi-tiered underlying topology among distributed cloud nodes, and builds defense belts to separate the segments based on the tiers. Then, this application controls the number of defense lines that networking traffic should traverse depending on its sources and destination, to enforce higher protection for important ICT assets. On the contrary, micro-segmentation can flexibly separate edge clouds depending on logical entities such as tenants and services, and focuses on applying different security policies to respective segments.

B. RELATED WORK ON SECURING MULTI-SITE EDGE CLUSTERS

To effectively protect multi-site edge clouds, various approaches have been proposed based on the concept of software-defined security. The authors of [9] proposed a framework to protect a centralized cloud by employing fog devices (corresponding to edge-boxes) as DDoS defense lines. [10] proposes autonomous security edge-boxes that can detect suspicious behaviors of end-things using ML algorithm for IDS. On the other hands, [11] and [12] suggest service function chaining-based security. The authors of [11] configure chains of containerized security functions in front of web servers for flow measurement and filtering. UniSec framework [12] implements physical security functions accelerated with FPGA (field programmable gate array)-based SmartNIC implementation. However, these works do not focus on the flexibility and scalability required to protect the complicated topology of multi-site cloud-native edge clouds effectively. Also, the visibility and semi-automated deployment features to cope with the increasing scale of distributed clusters are not clearly proposed in the work. On the other hand, SDSec [13] proposes a hierarchical architecture with protocols to effectively manage security rules across multi-site edge-boxes. Even though this work has similar focuses in managing rules of security functions, its targets are limited to physical and virtual boxes. Also, it does not fully address visibility for edge-cloud security.

Meanwhile, Linux eBPF and XDP, which have strong advantages in terms of usability, flexibility, lightweight, and performance, are widely used as an alternative technology of Linux internal security features such as iptables and traffic control [14], [15], [27]. In addition to example use cases provided by XDP [29], security orchestration actively leverages eBPF and XDP for securing distributed edge-cloud clusters. [16] employs eBPF-based functions to collect system-related information for security-oriented visibility to cover up to L7 networking traffic filtering. The proposed technology combines system visibility data of users, processes, and application containers measured by eBPF programs, with networking visibility data measured by *ntopng*. The authors of [17] replace iptables-based filtering functions with XDP

programs in the existing DDoS mitigation system, GateBot, for performance benefits and flexibility. When compared to these works, SmartX Multi-Sec attempts to propose a systematic flow-centric visibility framework with collective DevSecOps automation features, allowing DevSecOps operators to easily repeat monitoring, visualization, and filtering networking-based threats for multi-site cloud-native edge clouds.

VI. CONCLUSION

This paper proposes the SmartX Multi-Sec framework that attempts to provide a systematic structure of flow-based visibility and collective DevSecOps features for protecting multi-site cloud-native edge clouds. The SmartX Multi-Sec framework abstracts the underlying networking topology among virtualized and containerized cloud nodes as a multi-tiered onion-ring-style topology to address the topology intricacy in multi-site cloud-native edge clouds. Based on the topology abstraction, the collective DevSecOps functions such as visibility center, provisioning center, and eBPF-based DevSecOps functions support cloud operators to monitor, visualize, and filter networking-based threats. The visibility center supports scalable flow-centric visibility that collects, stores, analyzes, and stages networking packet traffic flowing over distributed edge clouds. Using the staged flow data, the visibility center provides 3D onion-ring visualization that can intuitively depict the multi-tiered topology together with the security status of respective networking ports. The provisioning center supports template-based deployment of DevSecOps functions for flexible flow capture and filtering on the targeted peering ports of physical, virtualized, and containerized cloud nodes. Linux eBPF/XDP-leveraged lightweight DevSecOps functions associated with physical, virtual, and container networking ports can directly generate flow features from networking packets in a flexible and resource-efficient way. We implemented the PoC-version of the SmartX Multi-Sec framework for K-ONE Playground for feasibility verification, which is our miniaturized testbed of multi-site cloud-native edge clouds. Based on an example scenario, we also showed the PoC-version of the SmartX Multi-Sec framework could successfully provide an intuitive 3D Onion-ring visualization, template-based easy function deployment, and resource-efficient flexible flow capture and filtering.

In the future, we will improve the data analysis module of SmartX Multi-Sec to address various attack scenarios by leveraging intelligent intrusion detection schemes based on machine learning and deep learning. In addition, with the flexibility of Linux eBPF, the framework will utilize heterogeneous events regarding computing and storage resources, to monitor and block system-oriented security threats.

REFERENCES

[1] H. Chang, A. Hari, S. Mukherjee, and T. V. Lakshman, "Bringing the cloud to the edge," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Toronto, ON, Canada, Apr. 2014, pp. 346–351.

[2] J. Pan and J. McElhannon, "Future edge cloud and edge computing for Internet of Things applications," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 439–449, Feb. 2018.

[3] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct./Dec. 2009.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, Helsinki, Finland, 2012, pp. 13–15.

[5] M. O. Keefe. (2018), *Edge Computing and the Cloud-Native Ecosystem*. Accessed: Sep. 1, 2020. [Online]. Available: https://thenewstack.io/edge-computing-and-the-cloud-native-ecosystem

[6] S. Kumar and J. Du. (2019). *KubeEdge, a Kubernetes Native Edge Computing Framework*. Accessed: Sep. 1, 2020. [Online]. Available: https://kubernetes.io/blog/2019/03/19/kubeedge-k8s-based-edge-intro

[7] CNCF Community. (2018). *CNCF Cloud Native Definition V1.0*. Accessed: Aug. 31, 2018. [Online]. Available: https://github.com/cncf/toc/blob/master/DEFINITION.md

[8] J.-S. Shin and J. Kim, "K-ONE playground: Reconfigurable clusters for a cloud-native testbed," *Electronics*, vol. 9, no. 5, p. 844, May 2020.

[9] Deepali and K. Bhushan, "DDoS attack defense framework for cloud using fog computing," in *Proc. 2nd IEEE Int. Conf. Recent Trends Electron., Inf. Commun. Technol. (RTEICT)*, Bangalore, India, May 2017, pp. 534–538.

[10] D. Zissis, "Intelligent security on the edge of the cloud," in *Proc. Int. Conf. Eng., Technol. Innov. (ICE/ITMC)*, Funchal, Portugal, Jun. 2017, pp. 1066–1070.

[11] E. Jalalpour, M. Ghaznavi, D. Migault, S. Preda, M. Pourzandi, and R. Boutaba, "A security orchestration system for CDN edge servers," in *Proc. 4th IEEE Conf. Netw. Softw. Workshops (NetSoft)*, Montreal, QC, Canada, Jun. 2018, pp. 46–54.

[12] J. Yan, L. Tang, J. Li, X. Yang, W. Quan, H. Chen, and Z. Sun, "UniSec: A unified security framework with SmartNIC acceleration in public cloud," in *Proc. ACM Turing Celebration Conf. China*, Chengdu, China, May 2019, pp. 1–6.

[13] M. Compastie, R. Badonnel, O. Festor, R. He, and M. Kassi-Lahlou, "A software-defined security strategy for supporting autonomic security enforcement in distributed cloud," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Luxembourg City, Luxembourg, Dec. 2016, pp. 464–467.

[14] M. Bertrone, S. Miano, F. Risso, and M. Tumolo, "Accelerating Linux security with eBPF iptables," in *Proc. ACM SIGCOMM Conf. Posters Demos*, Aug. 2018, pp. 108–110.

[15] J. Corbet. (2018). *BPF Comes to Firewalls*. Accessed: Sep. 1, 2020. [Online]. Available: https://lwn.net/Articles/747551

[16] L. Deri, S. Sabella, and S. Mainardi, "Combining system visibility and security using eBPF," in *Proc. Italian Conf. Cybersecur.*, Pisa, Italy, Feb. 2019, pp. 50–62.

[17] G. Bertin, "XDP in practice: Integrating XDP into our DDoS mitigation pipeline," in *Proc. Tech. Conf. Linux Netw. (Netdev)*, Apr. 2017, pp. 1–5.

[18] M. Usman and J. Kim, "SmartX multi-view visibility framework for unified monitoring of SDN-enabled multisite clouds," *Trans. Emerg. Telecommun. Technol.*, Dec. 2019, Art. no. e3819. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/ett.3819

[19] M. Usman, A. C. Risdianto, J. Han, and J. Kim, "Interactive visualization of SDN-enabled multisite cloud playgrounds leveraging SmartX multiview visibility framework," *Comput. J.*, vol. 62, no. 6, pp. 838–854, Jun. 2019.

[20] M. Usman, M. A. Rathore, and J. Kim, "SmartX multi-view visibility framework with flow-centric visibility for SDN-enabled multisite cloud playground," *Appl. Sci.*, vol. 9, no. 10, p. 2045, May 2019.

[21] J. -S. Shin and J. Kim, "Multi-layer onion-ring visualization of distributed clusters for SmartX multiview visibility and security," presented at the IEEE Symp. Vis. Cyber Secur. (VizSec), Oct. 2018. [Online]. Available: https://vizsec.org/files/2018/Shin_Poster.pdf

[22] M. Usman, N. T. Manh, and J. Kim, "Multi-belt onion-ring visualization of OF@TEIN testbed for SmartX multi-view visibility," in *Proc. 9th Int. Workshop Comput. Sci. Eng. (WCSE)*, Yangon, Myanmar, 2019, pp. 6–10.

[23] J. Kim and T. Nam, "Cluster visualization device," U.S. Patent 16 629 299, Mar. 13, 2018.

[24] A. C. Risdianto, M. Usman, and J. Kim, "SmartX box: Virtualized hyper-converged resources for building an affordable playground," *Electronics*, vol. 8, no. 11, p. 1242, Oct. 2019.

[25] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proc. Mil. Commun. Inf. Syst. Conf. (MilCIS)*, Canberra, ACT, Australia, Nov. 2015, pp. 1–6.
[26] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy (ICISSP)*, Funchal, Portugal, 2018, pp. 1–8.
[27] B. Gregg. (2016). *Linux Extended BPF (eBPF) Tracing Tools*. Accessed: Aug. 31, 2020. [Online]. Available: http://www.brendangregg.com/ebpf.html
[28] J. Schulist, D. Borkmann, and A. Starovoitov. *Linux Socket Filtering Aka Berkeley Packet Filter (BPF)*. Accessed: Aug. 31, 2018. [Online]. Available: https://www.kernel.org/doc/Documentation/networking/filter.txt
[29] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "The eXpress data path: Fast programmable packet processing in the operating system kernel," in *Proc. 14th Int. Conf. Emerg. Netw. Exp. Technol. (CoNext)*, Heraklion, Greece, Dec. 2018, pp. 54–66.
[30] J.-S. Shin and J. Kim, "Template-based automation with distributed secure provisioning installer for remote cloud boxes," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Jeju, South Korea, Oct. 2016, pp. 33–35.
[31] D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak, and G. Carle, "Performance implications of packet filtering with Linux eBPF," in *Proc. 30th Int. Teletraffic Congr. (ITC)*, Vienna, Austria, Sep. 2018, pp. 209–217.
[32] D. Kuipers and M. Fabro, "Control systems cyber security: Defense in depth strategies," Idaho Nat. Lab., Idaho Falls, ID, USA, Tech. Rep. INL/EXT-06-11478, May 2006.

JUN-SIK SHIN received the B.S. degree in information and computing engineering from Ajou University, Suwon, South Korea, in 2007, and the M.S. degree from the School of Information and Communication, Gwangju Institute of Science and Technology (GIST), Gwangju, South Korea, in 2014, where he is currently pursuing the Ph.D. degree with the School of Electrical Engineering and Computing Science.

His research interests include multi-site composable edge clouds and infrastructure automation.

JONGWON KIM (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in control and instrumentation engineering from Seoul National University, Seoul, South Korea, in 1987, 1989, and 1994, respectively. From 1994 to 2001, he was a Faculty Member with Kongju National University, Gongju, South Korea, and the University of Southern California, Los Angeles, CA, USA. In 2001, he joined Gwangju Institute of Science and Technology (GIST), Gwangju, South Korea, where he is currently a Full Professor. Since 2008, he has been directing GIST Super Computing Center. Since 2019, he has been the Dean of GIST AI Graduate School. He is also leading Networked Computing Systems Laboratory, where he is involved in dynamic and resource-aware composition of media-centric service employing programmable/virtualized computing/networking resources. His recent research interests include agile and visible p+v+c function-leveraged composition of SmartX IoT-cloud services employing programmable/sliced/hyper-converged (computing/storage/networking) resources.