Received 19 July 2024, accepted 15 August 2024, date of publication 20 August 2024, date of current version 28 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3446535

RESEARCH ARTICLE

BotFence: A Framework for Network-Enriched Botnet Detection and Response With SmartNICs

HYUNMIN SEO[1], SEUNGWON SHIN[1], (Member, IEEE), AND SEUNGSOO LEE[2]
[1]School of Electrical Engineering, KAIST, Yuseong-gu, Daejeon 34141, Republic of Korea
[2]Department of Computer Science and Engineering, Incheon National University, Yeonsu-gu, Incheon 22012, Republic of Korea

Corresponding author: Seungsoo Lee (seungsoo@inu.ac.kr)

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) funded by Korean Government [Ministry of Science and ICT (MSIT)] through the Development of Darkweb Hidden Service Identification and Real IP Trace Technology under Grant RS-2022-II220740.

ABSTRACT The scale of botnet attacks is on the rise, yet traditional network security systems are inadequate to effectively respond to these threats, primarily due to high false positive rates and the extensive manpower required for analysis. In contrast, the cutting-edge method of intrusion detection, known as provenance-based analysis, offers a novel paradigm by establishing causality between host events for meticulous examination. Nonetheless, this method faces challenges in analyzing the payload of network packets, which contains critical attack information resides, due to performance efficiency constraints from packet inspection. To address these challenges, we introduce *BotFence*, a pioneering approach that integrates payload inspection of network packets with provenance-based analysis to enhance botnet intrusion detection and response. Notably, our system leverages SmartNICs to minimize the impact on network performance. Our system initially gathers and analyzes events within the host system, representing them as Tactics, Techniques, and Procedures (TTP). Concurrently, it collects and scrutinizes the network packets associated with these events, integrating the relationships between these TTPs and the collected network data into a Network-enhanced Threat Provenance Graph (NTPG) model that we devised. Consequently, our system provides a comprehensive security analysis of the network with minimal overhead. Demonstrations with complex attack scenarios show that *BotFence* successfully identifies and mitigates automated botnet infection in real time, analyzing more than 99. 9% host events in 1 ms, without degrading network performance.

INDEX TERMS Botnet, endpoint detection and response, programmable dataplane, network security.

I. INTRODUCTION

The World Economic Forum's Global Risks Report 2023 [1] identifies cyberattacks on critical infrastructure as a swiftly escalating global risk, projecting that cybercrime costs will ascend to $10.5 trillion annually by 2025. The prevalence of botnet attacks is also escalating, with a documented annual increase of 25% [2]. A botnet constitutes a network of compromised computers that are collectively orchestrated to execute coordinated cyber-attacks. These networks exhibit significant offensive capabilities and are instrumental in conducting a variety of highly disruptive operations, including Distributed Denial of Service (DDoS) attacks, spam dissemination, and phishing campaigns, all managed through a centralized command and control infrastructure. Typically, bot-infected systems operate without the user's awareness, complicating both detection and eradication processes, thus prolonging the duration of the attack. Furthermore, botnets facilitate malicious activities targeting infected computers and networks, such as exfiltrating sensitive data (e.g., personal or login information), deploying ransomware to extort funds, and exploiting system resources for cryptojacking.

To counter such botnet infection, there have been numerous security solutions utilizing logs from host systems as a

The associate editor coordinating the review of this manuscript and approving it for publication was S. K. Hafizul Islam.

means of detecting intrusions. In traditional attack detection systems, such as Endpoint Detection and Response (EDR) [3] and Security Information and Event Management (SIEM) [4] systems, the malicious activities was predicated on the collection and analysis of various system logs occurring within individual hosts. However, this approach induces significant challenges in the identifying malicious behavior [5]. (*i*) The necessity to monitor and analyze a vast array of events for intrusion attack detection demands extensive data processing and analytical capabilities. (*ii*) The protracted nature of attacks complicates the continuous storage and management of all pertinent logs and data. (*iii*) The resemblance of attacker actions to legitimate user activities, making distinction arduous and leading to a high rate of false positives. Given these multifaceted challenges, the field of intrusion attack detection and response has evolved beyond conventional log analysis.

In response to those challenges, the field has seen the rise of provenance-based host behavior analysis as a preferable alternative to traditional security frameworks [5], [6], [7], [8], [9], [10], [11]. This method involves the collection of information regarding the behavior of the host system, including the creation and termination of processes, file system interactions, and network activities, categorized as host events. The analysis of the causality among these events aids in the identification of attack signatures. Through the use of a provenance graph to depict the causal connections between host behaviors, the technique offers an analysis that includes a definitive explanation of the attack scenario and a detailed investigation of the threat's root cause.

However, this method still possesses drawbacks, particularly in its limited efficacy in detecting network-related malicious events. This limitation arises because the provenance-based approach is predominantly reliant on host event data and only garners limited information from network packets [12]. Considering that botnets are constructed and orchestrated via network connections, network packets provide significant indications of malicious activities. The payload of these packets is likely to carry distinct signatures of the attack, such as Command and Control (C&C) communications and malicious code [13], making the analysis of packet payloads a potent method for identifying malicious activity. Nevertheless, traditional host-based approaches, including provenance-based analysis, often overlook the potential of network payload inspection, primarily due to concerns over performance degradation and resource consumption [14]. Moreover, adversaries frequently employ payload encryption techniques to obscure their actions, thereby complicating the process of direct packet inspection [15], [16]. Consequently, such systems are compelled to detect the attacks by analyzing the relationships between host events rather than through direct observation of network packets. Also, this methodology necessitates the aggregation of host events on a central server for subsequent analysis, which incurs significant latency, thereby hindering immediate response to attacks.

Thus, we propose *BotFence*, a novel system architecture designed to concurrently monitor host and network environments by leveraging the capabilities of SmartNICs [17], as programmable dataplanes, can be seamlessly integrated into hosts to perform a plethora of operations pertinent to network packet management. These devices are equipped with packet-processing hardware accelerators and facilitate diverse packet steering mechanisms without compromising network throughput. Our system harnesses the advantages offered by SmartNIC for accurate packet inspection. Furthermore, recent advancements in SmartNIC technology have seen the integration of general-purpose processors, such as ARM cores, into the NICs. *BotFence* capitalizes on this feature to delegate the task of analyzing host events to the SmartNIC, thereby offloading significant computational workloads from the host CPU.

BotFence is designed to aggregate host events and distill them into high-level Tactics, Techniques, and Procedures (TTPs) [18]. It constructs a provenance graph that delineates the causal relationships among TTPs and enhances it by appending network information, resulting in the Network Enhanced Threat Provenance Graph (NTPG). To achieve this, *BotFence* concurrently scrutinizes network packets and correlates these findings with host events, facilitating the timely detection and mitigation of botnet attacks. In addition, our system leverages the specialized Deep Packet Inspection (DPI) engine, which ensures line-rate network throughput with segregation of the inspection rules, thereby minimizing performance degradation. *BotFence* demonstrates a high processing capability, handling over 99.9% of events within a 1-millisecond timeframe, thereby supporting real-time detection and response mechanisms.

In summary, we make the following contributions:

- We introduce *BotFence*, a real-time botnet intrusion detection and response system augmented by SmartNICs. This system reduces the workload related to the attack detection and analysis by integrating comprehensive packet inspection seamlessly.
- We design the Network enhanced Threat Provenance Graph (NTPG), an innovative methodology for performing provenance-based botnet intrusion analysis, enriched with network packet information.
- We evaluate *BotFence* using a meticulously designed attack scenario, demonstrating the system's proficiency in real-time attack identification without impairing network performance.

The rest of this paper is organized as follows: Section II provides the background in botnet attacks and data provenances. Section IV describes the motivation for our work. The overall design of *BotFence* and its detail are presented in Section V and Section VI respectively. The implementation of *BotFence* and its evaluation results are summarized in Section VII. Section VIII discusses the limitations of the current design. Section III reviews previous studies and their limitations. Finally, we conclude this paper in Section IX.

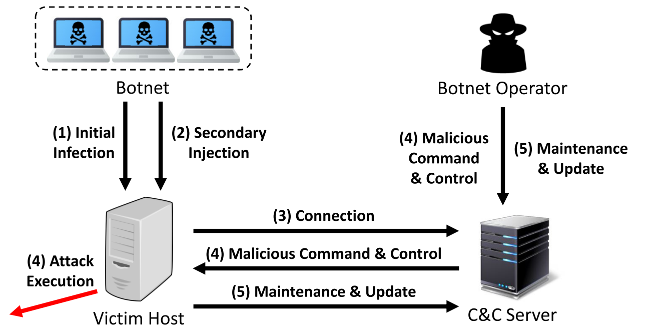


FIGURE 1. **Lifecycle of Botnet.**

II. BACKGROUND

A. CYBER ATTACK WITH BOTNET

A botnet is a system in which an attacker links multiple computers into a single network and remotely controls them to execute attacks. To configure these networks and infect as many hosts as possible, attackers often utilize automated malware that replicates itself and spreads across the network, thereby expanding the scope of infection and taking over the network to form a botnet. Figure 1 illustrates the life cycle of a botnet. Initially, the botnet identifies vulnerable hosts and performs the initial malware injection. It exploits host vulnerabilities or uses phishing emails containing malicious attachments or links to infect host systems and establish a foothold. Subsequently, It downloads and executes malicious codes to integrate the infected host into the botnet. Once a host becomes a member of the botnet, it connects to the Command and Control (C&C) server set up by the attacker to receive remote commands. The host establishes a communication channel and waits for instructions from the attacker. To carry out an attack, the botnet operator issues commands through the C&C server, enabling various malicious activities. Botnets constructed in this manner are typically employed for cybercrime, engaging in activities such as Distributed Denial of Service (DDoS) attacks, sending spam emails, data theft, and generating clicks on illegal advertisements.

Botnets target various hosts, including PCs, servers, and IoT devices, automatically identifying and infecting vulnerable devices without human intervention. Mirai [19], the most well-known botnet, specifically targets IoT devices, obtaining administrator privileges through brute force attacks. It then downloads and executes the malicious payload from the C&C server, integrating the infected device into the botnet. Botnets targeting common hosts, such as Emotet [20], Conficker [21], Zbot [22], and Necurs [23], primarily propagate through phishing emails containing malicious attachments or links. They gain control of the system by executing macros, deploying Trojan horses, or exploiting security vulnerabilities, subsequently downloading malicious payloads to incorporate the infected host into the botnet.

Additionally, botnets employ various network techniques such as encryption and obfuscation, camouflaging into network traffic, and domain generation algorithms to evade detection. Since they must receive commands from the C&C server, maintaining communication with the C&C is essential. Consequently, botnets use regular web traffic, such as HTTPS, to conceal command messages and employ encryption and obfuscation to render the payload indiscernible. Additionally, to obscure the address of the C&C server, techniques like DNS fast flux [24] or domain generation algorithms are used to hide the domain and IP address utilized in the commands. These attack detection and evasion techniques significantly complicate the detection of botnets.

B. BOTNET DETECTION WITH DATA PROVENANCE

During the process of botnet intrusion, attackers inevitably generate numerous indicators in the form of host events. These events serve as a record of a host's activities, encapsulating a range of actions from process creation and termination to file system access and network socket creation. The anomalous activities indicative of an attacker's malicious endeavors often manifest distinctly when compared to the baseline behavior of a typical host, enabling the detection of intrusions through the monitoring of these host events. Traditional host log analysis-driven detection frameworks, such as Endpoint Detection and Response(EDR) and Security Information and Event Management(SIEM) systems, ingest these host events in the form of logs for subsequent analysis [3], [4]. Their primary objective is to identify indicators of compromise within the logs related to specific host services. Furthermore, these systems aggregate and scrutinize multiple logs to discern patterns of malicious activity.

However, traditional methods often overlook the crucial correlation between logs, rendering the detection of complex attack patterns increasingly challenging due to the prevalence of false positives and the substantial human effort required for analysis [5], [7]. In response to these limitations, the application of data provenance in enhancing intrusion attack detection capabilities is being explored. Provenance-based intrusion detection approaches elucidate the causality of security incidents by examining the relationships between system executions [25]. This method involves the collection of host event data to explore the precedence relationships between events, facilitating the analysis of malicious behavior through graphical representation. With attack behaviors depicted graphically, it enables the identification of malicious activities through the examination of event sequences that have been classified as malevolent or by contrasting them against patterns of normal operations.

It also becomes feasible to trace the origins of an attack by navigating backwards in the graph. This enables the identification and fortification of vulnerabilities that could precipitate further malicious activities. Conversely, forward tracing within the graph aids in discerning the objectives of the attackers, allowing for targeted security enhancements. Furthermore, this approach involves the exploration of event correlations across multiple hosts, thereby enabling a holistic comprehension of their interdependencies and potential security implications. By focusing on the behavioral analysis of hosts, data provenance analysis surpasses traditional log analysis methods in its capacity for early detection of cyber threats, thus offering a significant advantage in the preemptive identification of malicious activities.

III. RELATED WORKS

Botnet infections deploy automated malware aimed at maximizing host infiltration and assimilating them into the botnet's network. Consequently, the methodologies employed in these attacks have evolved to become increasingly sophisticated and covert [19], [26], [27]. In response to this evolving threat landscape, cybersecurity systems have evolved, broadly categorizing into network-based and host-based security solutions, contingent upon the domain of attack detection.

Network-based security systems, such as Intrusion Detection Systems (IDS) [28] and Intrusion Prevention Systems (IPS) [29], engage in the monitoring and analysis of network traffic to identify potential threats. These systems scrutinize packets traversing the network, employing signature-based detection to identify known attack patterns and anomaly-based detection to flag irregular traffic behaviors [30]. Such approaches enable the real-time analysis of voluminous network data streams, facilitating the early identification of intrusion attempts. Nonetheless, these mechanisms exhibit limitations in decrypting encrypted traffic and in identifying threats that emanate from within the internal network or from adversaries who have successfully breached network defenses [31]. Conversely, host-based security solutions focus on monitoring and analyzing activities on specific hosts, including computers or servers. Solutions within this category, such as Endpoint Detection and Response (EDR) [3] and Security Information and Event Management (SIEM) [4] systems, are designed to continuously monitor and document activities on individual endpoints, identifying anomalous behaviors or known threat signatures. However, these solutions impose computational burdens on host systems, potentially impacting performance [12]. Moreover, their reactive nature means that detection occurs post-incident, potentially missing opportunities for preemptive attack mitigation.

Recent advancements in cybersecurity research have led to the development of methodologies that adeptly organize and analyze logs generated by host systems and security frameworks, employing data provenance analysis to elucidate causal relationships between events for enhanced intrusion detection. Notably, SLEUTH [6] pioneered the representation of attacks on enterprise hosts via provenance graphs, facilitating attack detection through tagging mechanisms, and MORSE [8] sought to diminish the incidence of false positive alerts by introducing a refined policy model. Similarly, HOLMES [7] leveraged data provenance analysis alongside Tactics, Techniques, and Procedures (TTP) to articulate attack scenarios at a granular level, thereby streamlining analytical processes. Similarly, RapSheet [5] constructed a TTP graph derived from logs aggregated by extant EDR systems, applying data provenance principles to enhance attack detection capabilities. These methodologies primarily concentrate on host-level events, endeavoring to identify intrusions through the implementation of alert minimization and alarm triage protocols.

Distinct from these approaches, which rely on establishing direct matching rules for TTP identification and attack detection, other studies have explored the detection of anomaly-based attacks. UNICORN [9], for instance, introduced a novel method for detecting prolonged attack by transforming the graph into sketches, which serves as a clustering feature, with outliers subsequently categorized as anomalies. ProveDetector [10] employed a probability density-based algorithm, known as the Local Outlier Factor, to identify stealth malware. Meanwhile, Poirot [11] approached attack detection as a graph pattern matching challenge, aligning the provenance graph with a Query Graph derived from Cyber Threat Intelligence, leveraging various machine learning techniques to discern data deviating from normative patterns as malicious.

However, these methodologies share a common limitation in their delayed response to detected threats, attributed to the protracted duration required for identifying genuine threats solely through host-level events. Additionally, the necessity of centralizing data collection for analysis compounds the latency, as the time involved in data transmission, central processing, and decision-making extends the interval before protective measures can be enacted, rendering the immediate mitigation of attacks nearly unfeasible.

On the other hand, *BotFence* enhances attack detection capabilities by synergizing the analysis of network packets with conventional host event-based methods, addressing the limitations inherent in solely relying on host events for identifying threats. By incorporating previously overlooked network packet information, our system achieves more precise identification of cyber threats, leveraging the untapped potential of network packet data to augment the botnet detection.

In addition, while aforementioned approaches focus on identifying the formation of botnets, considerable effort has also been made to detect and mitigate attacks post-formation. Botnets, once established, frequently engage in DDoS attacks with the objective of incapacitating target networks. Consequently, it is crucial to identify and thwart signs of DDoS attacks before the traffic overwhelms the network. Extensive research has been conducted to predict real-time fluctuations in attack traffic volume [32], [33], [34]. Our study aims to detect attacks during the botnet formation stage, and can be effectively integrated with existing post-formation attack mitigation methodologies to improve overall defense against botnet-related threats.

IV. MOTIVATION

Despite the efficacy of data provenance analysis in detecting intrusion attack of botnet infection, it exhibits certain limitations. Systems employing data provenance analysis predominantly collect host event data through conventional audit frameworks, yet they eschew the direct monitoring of network packet payload primarily due to concerns over performance degradation. The necessity for deep packet inspection (DPI) to obtain packet data from hosts introduces substantial computational overhead. DPI entails rigorous regular expression matching processes, which, when executed in software, lead to significant CPU resource consumption [14]. This, in turn, detrimentally impacts not only network performance but also the overall system efficiency. Consequently, to circumvent these limitations, systems opt to collect indirect network information through the acquisition of headers or analysis of traffic patterns. However, this approach is susceptible to evasion by adversaries, who may alter IP addresses or port numbers to bypass header-based detection mechanisms, and it is also prone to a higher rate of false positives associated with traffic pattern analysis.

Nevertheless, the integration of DPI with host events is crucial, as broadening the scope of data collection to encompass detailed network information significantly enhances the effectiveness of attack identification processes. One critical purpose of packet inspection is the detection of Command and Control (C&C) servers, which are crucial for facilitating communication between attackers and compromised systems [13]. A deeper understanding of the attack's commands, characteristics, and goals can be obtained through the direct analysis of communications with these servers by examining packet payloads. This analysis allows for the detection of malicious code or links, revealing attempts by attackers to spread malware or infiltrate systems. However, attackers frequently use encrypted communications to conceal their traffic, making traditional packet payload analysis methods, such as IDS, ineffective [15], [16]. The monitoring of attacks with encrypted traffics becomes significantly more difficult as messages are hidden behind the ciphertext. To analyze such network traffic with traditional approach, it is imperative to implement techniques that initially decrypt the packets, subsequently analyze the payload, and thereafter re-encrypt it whilst in transit through the network infrastructure [35]. Employing these methods requires additional resources for redundant decryption and encryption, which may further reduce network performance.

A. MOTIVATING EXAMPLE

Consider a scenario where an attacker successfully deploys malware on a host and subsequently attempts to establish a connection with the C&C server, as depicted in Figure 2. Traditional security systems, which primarily focus on collecting host events, conduct a limited examination of network packets, typically verifying connections based solely on specific IP addresses or ports. However, when attackers employ DNS Fast Flux [24] techniques to frequently change the IP addresses or leverage VPNs to obscure their IP, identifying the attack becomes significantly more challenging. Moreover, attackers often mask their activities by utilizing encrypted HTTPS web traffic, rendering the payload contents opaque. Consequently, if attack detection relies solely on host events, all HTTPS connections would need to be treated as potential threats. Yet, not all connections are inherently malicious, and distinguishing between benign and malicious connections necessitates an understanding of the relationship between host events and network sessions deemed malicious. This requirement makes the process inefficient for prompt actions.

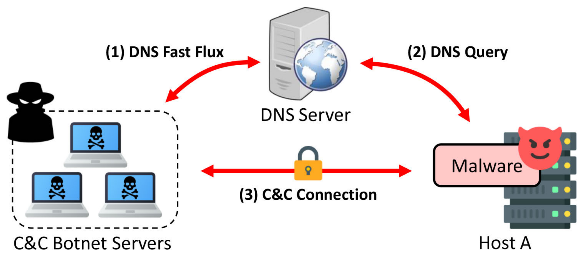


FIGURE 2. Example scenario of C&C session establishment in botnet attack. The attacker hide its IP address by DNS fast flux and encrypt C&C message.

Consequently, conducting immediate and detailed payload analysis of network packets is crucial for the detection of botnet intrusion attacks. This process should incur the lowest possible overheads to ensure efficiency. By doing so, it provides a wealth of critical information essential for enhancing the data provenance approach, enabling a more effective identification and understanding of such attacks.

V. SYSTEM OVERVIEW

A. DESIGN CONSIDERATION

Network packets are crucial for botnet detection, necessitating the integration of packet analysis with data provenance analysis. Therefore, we propose the integration with SmartNICs to effectively conduct the network packet inspection in enhancing data provenance-based botnet infection analysis. Specifically, the workload of data provenance-based attack analysis is offloaded into SmartNIC to enable monitoring of host events with detailed and real-time examination of packet payloads using Deep Packet Inspection (DPI) technology. This approach seeks to optimize the correlation between network packet data and host events, thereby impeding the ability of malicious processes to effectively interact with the network. Thus, we first derive the following design considerations that a system should adhere to enable real-time monitoring of host events and detailed examination of packet payloads with minimal overheads.

First, BotFence should monitor malicious process activities with its network packet payload and formulate network rules capable of preempting attacks. This begins by collecting host activities through the aggregation of host events, which are then offloaded for in-depth analysis. This process involves characterizing host events, subsequently translating

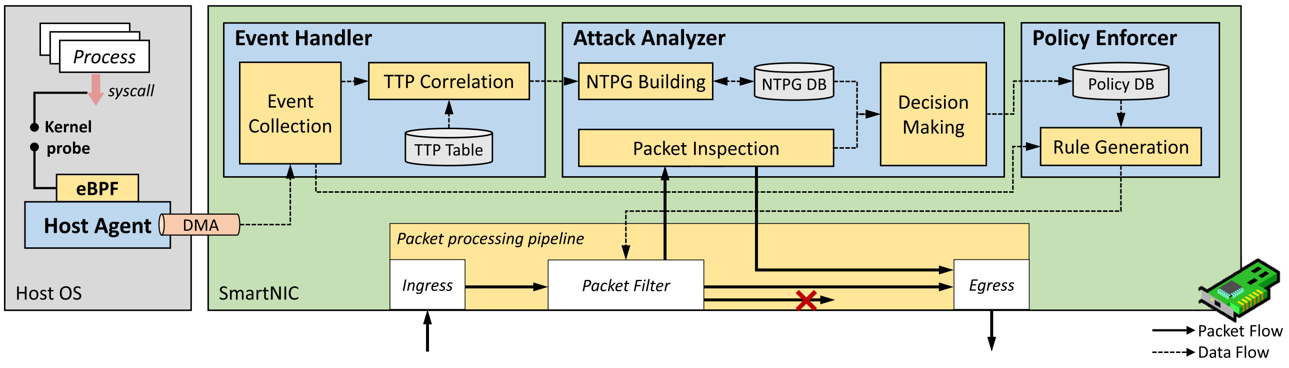


FIGURE 3. The overall architecture and workflow of *BotFence* with four key components: (*i*) Host Agent, (*ii*) Event Handler, (*ii*) Attack Analyzer, and (*iv*) Policy Enforcer.

them into Tactics, Techniques, and Procedures (TTPs) for high-level observation. Concurrently, our system examines packets entering and exiting the host to assess their potential maliciousness. The integration of host event data with network packet inspection culminates in the creation of the Network enhanced Threat Providence Graph (NTPG), laying the groundwork for identifying malicious activities.

Second, BotFence should significantly enhance the efficacy of packet monitoring and encryption handling to minimize the impact on the network performance. All packets entering and exiting the host are compulsorily processed through the SmartNIC, our system facilitates such architecture to comprehensively monitor the traffic of the host. This approach supports the dynamic steering and blocking of packets, playing a pivotal role in preventing potential attacks. Furthermore, our system inspects packet payloads subsequent to decryption by utilizing packet encryption offloading. The techniques such as kernel TLS (kTLS) is designed to reduce copy overhead by performing TLS encryption and decryption directly within the kernel [36], a process that can also be offloaded to hardware components. By leveraging this functionality within the host operating system, our system can inspect payloads of unencrypted packets. These decrypted plaintext packets can then be delivered to the host OS, ensuring seamless security integration without degradation of performance.

B. THREAT MODEL AND ASSUMPTIONS

This study focuses on detecting botnet construction of botnets within an enterprise environment consisting of multiple hosts. A ‘host’ is defined as any general computing device or server within a network, which has the capability to interconnect with other hosts internally and communicate with external networks. This interconnectivity facilitates resource sharing and external communication. The construction of botnets is initiated through automated malware that infiltrates the host via system vulnerabilities or phishing methods. Upon initial penetration, the malware seeks to propagate to additional hosts, with the objective of dominating and controlling the entire target network, thereby integrating it into the botnet. These attacks typically employ network-based strategies due to the lack of physical access to the hosts.

Our objective is to detect and prevent the formation of botnets. Consequently, we do not consider detecting attacks that occur post-botnet creation (such as DDoS). The deployment of our system involves using a SmartNIC, which interfaces with the host through a PCIe channel. Notably, in restricted mode, the host is prevented from accessing the SmartNIC control channel. This restricted mode can only be configured during the boot process of the SmartNIC, after which all control channels, except those requiring physical access, are deactivated.

C. SYSTEM ARCHITECTURE

Figure 3 illustrates the overall architecture and workflow of *BotFence* with four key modules. The pathway facilitates data transfer and packet movement among the different modules to monitor host events, amalgamate packet information to construct the Network enhanced Threat Providence Graph (NTPG), and identify attacks. Especially, our system executes attack detection by integrating network packet inspection with the collection of host events via kernel probe, with minimal overheads. This approach is complemented by the analysis of events through data provenance analysis, offering a comprehensive method for identifying and mitigating cyber threats. To achieve this, our system conducts three core operations as follows.

1) HOST EVENT HANDLING

BotFence initiates this process at the stage of host event collection. This is achieved through the deployment of the **host agent** module, which utilizes Extended Berkeley Packet Filter (eBPF) technology to interface with the host kernel. The primary function of this agent is to intercept host events, which are then transmitted to a SmartNIC via the Direct Memory Access (DMA) channel. Upon reception, the **event handler** module processes these host events, transforming them into TTPs based on predefined criteria within a TTP table. The transformation allows a single TTP to reflect a singular event or an aggregation of multiple events.

TABLE 1. The three different categories of the host events monitored by the host agent.

Category	*Operation*	*Data*
Process	Create, Terminate	PID, PPID, System call, Arguments
File system	Create, Read, Write, Delete	PID, Filename
Network	Packet Send / Receive	PID, 5-tuple

2) ATTACK ANALYSIS

The next phase involves the **attack analyzer** module, which integrates the processed TTPs with network packet data to determine the presence of cyber-attacks. This module interrelates TTPs based on their origin and causal relationship. Concurrently, it examines incoming and outgoing packets of the hosts to identify potentially malicious activities. The attack analyzer formulates an attack scenario by synthesizing the sequence of TTPs with insights from network packet analysis to construct NTPG, thereby facilitating comprehensive attack detection.

3) SECURITY POLICY ENFORCEMENT

The final module, the **policy enforcer**, operationalizes the attack determinations made by the attack analyzer. It generates network rules aimed at either preventing the activities of malicious processes or monitoring for suspicious behaviors. These rules are integrated into the *packet processing pipeline*, equipped to execute actions such as blocking, inspecting, or collectively offloading network packets based on the established rules. The decisions to direct packets to the host or to execute predefined actions are governed by these rules, enabling effective network security management.

VI. SYSTEM DETAIL

A. HOST EVENT HANDLING

Within a host environment, numerous host events relevant to the botnet operations occur. The host agent, installed within the host operating system, is tasked with aggregating these events. It utilizes the eBPF to collect host events by hooking into the kernel probe. This interception enables the collection of system calls raised by processes along with their parameters. For example, the action of a parent process initiating a child process is traced through the interception of a kernel probe of the `sys_clone()` system call, revealing the pid of the child process. A compendium of these collected events is cataloged in Table 1. The *host events* gathered by the agent are then transmitted to event handler via the DMA channel, facilitating offloaded analysis.

The event handler performs dual functions; aggregating events and correlating TTP. For aggregation, this module receives host events from the host agent through the DMA channel and organizes these events into a standardized format, referred to as *feeds*. A unique feed is generated for each host event, which encapsulates the identifier of the process (pid) responsible for the event, the category, the specific operation executed, and an array of variables (arguments) associated with the event. These feeds are crucial for conducting TTP correlation within the event handler and enable the delivery of network events to additional modules within *BotFence* for further integrated analysis.

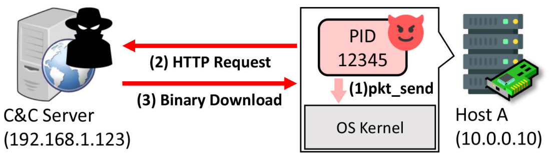


FIGURE 4. The botnet malware installed in the host tries to connect C&C server via HTTP protocol.

For the TTP correlation, the event handler compares the processed feed against a pre-established set of generation rules in TTP table. The rules are formulated with reference to the MITRE ATT&CK TTP database [37] and established a priori to the activation of the system. With rules, event handler represents the feed, which is the low-level host event as a corresponding high-level TTP. For instance, if a process performs a read operation on the contents of the `/etc/passwd` file followed by the creation of a network socket, this sequence of events can be recognized as a TTP that signifies the extraction of local file data (T1005 - Data from Local System) [38]. Once classified into TTP formats, the host events are forwarded to the attack analysis module for provenance analysis, thereby enhancing the accuracy of threat detection and characterization.

Figure 4 illustrates a scenario that demonstrates how our system captures and processes events from the host. Initially, malware infiltrates Host A through an unidentified entry point, installs itself, and becomes operational with PID 12345. The malware then attempts to establish a connection with a Command and Control (C&C) server to download additional malicious code. To achieve this, the malware on the host sends an HTTP request to the C&C server using port 80. This process involves the malware transmitting the payload to the kernel via the `pkt_send` system call (1), which constructs a packet containing the HTTP request and forwards it to the C&C server (2). Upon receiving the request, the C&C server responds by sending back a binary file laden with malicious code to Host A (3).

To prevent such attack scenario, we assume there exists a TTP generation rule[1] as shown in Figure 5(a). According to this rule, a TTP is generated with specific PID when a host event involves a process initiating a `pkt_send` event within the network category and targeting port 80; this is classified under T1071.001 (Application Layer Protocol: Web Protocols) [39]. The malware starts running and the host agent collects the pkt_send system call initiated by the malicious process (PID: 12345) via a kernel probe. This system call, along with the PID and its arguments, is relayed to the event handler through the DMA channel, initiating the

[1] We refered RapSheet [5] for the format of the rule.

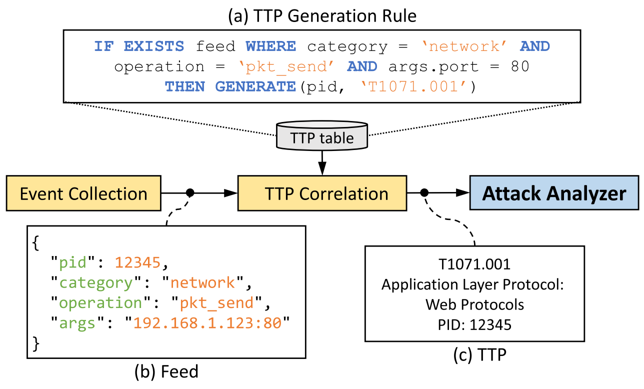


FIGURE 5. The transformation of the host event received from the agent into a TTP.

data processing for attack analysis. The event is converted into a standardized feed ((b) in Figure 5) and forwarded to the TTP correlation submodule. This feed indicates that PID 12345 initiated a pkt_send event within the network category, targeting the address 192.168.1.123:80. Upon receiving the relevant feed, the TTP correlation submodule applies the TTP generation rule to the feed. Figure 5(c) displays the resultant TTP node, indicating that process PID 12345 has been implicated in generating the T1071.001 TTP node. This node is then conveyed to the attack analyzer. The delineated TTP corresponds to event (2) in Figure 4, specifically, the transmission of an HTTP request to the C&C server. In addition, event (3) in Figure 4, involving the download of a malicious binary, can be processed and transformed to another TTP in a similar manner.

B. ATTACK ANALYSIS

Within the attack analyzer module, an NTPG is constructed to elucidate the causal relationships among TTPs with relevant network activities. The NTPG is first organized by sequencing TTPs chronologically, based on the PID, thereby facilitating an intuitive understanding of the temporal precedence of events orchestrated by a specific process. Subsequently, the attack analyzer inserts the constructed NTPG into the database. Upon the emergence of a new TTP node from the event handler, the relevant NTPG is queried from the database to aid in the dynamic assembly of the graph, enabling a comprehensive analysis of attack vectors.

Basically, NTPGs are managed based on PIDs, so it is critical to thoroughly track the lineage between parent-child processes. For example, when a malicious process initiates a child process that, in turn, triggers an event, the PID recorded at the event may differ from that of the initiating malicious process. Thus, relying exclusively on PID for TTP association presents a challenge, as it may not link events emanating from the child process. To address this issue, it is essential to ascertain the ancestor process information via the parent PID (PPID). The inception of a child process typically involves a system call, such as `clone`, enabling the identification of both the child process's PID and its PPID.

Algorithm 1 NTPG generation
```
Inputs : Feed $f$; Packet inspection result $d$
Output: Network enhanced Threat Provenance Graph
        $NTPG$
Data: List of TTP generation rules $List_{rule}$
foreach $rule \in List_{rule}$ do
    $TTP \leftarrow MatchRule(f, rule)$;
end
$NTPG \leftarrow \phi$;
if $TTP \neq \phi$ then
    $pid \leftarrow GetPid(f)$;
    $category \leftarrow GetCategory(f)$;
    $NTPG \leftarrow GetNTPG(pid)$;
    if $category = network$ then
        $ft \leftarrow Get5tuple(f)$, 5-tuple of feed;
        $dt \leftarrow Get5tuple(d)$, 5-tuple from DPI result;
        if $ft = dt$ then
            $N \leftarrow d$, network node;
            $TTP \leftarrow TTP \cup N$;
        end
    end
    $NTPG \leftarrow NTPG \cup TTP$;
end
```

TABLE 2. Severity and numeric score of TTP.

Severity	Low	Moderate	High	Critical
Score	2.0	6.0	8.0	10.0

To enhance the attack analysis with completion of NTPG, our system incorporates the inspection of network packets, which is not extensively explored in traditional provenance-based attack analysis. The attack analyzer examines the payload of packets using DPI technique to identify the presence of malicious content. Moreover, the attack analyzer utilizes the DPI engine, optimized for DPI tasks, enabling thorough inspections without impeding network performance. The inline packet examination is conducted within the packet processing pipeline; upon detection of a malicious payload, the attack analyzer establishes correlations with TTPs by leveraging the five-tuple information extracted from the packet, thereby beginning the decision-making process. Algorithm 1 illustrates the whole sequence of processes involved in the NTPG creation, detailing the generation of TTP from the feed within the event handler, correlation of TTPs with PID and incorporation of network packet inspection results.

Our system evaluates the malevolence of process activities by scoring the constructed provenance graph and integrating the outcomes of malicious message assessments conducted through packet inspection. The scoring of each TTP within the graph aims to quantify the severity of the NTPG, referencing the Common Attack Pattern Enumeration and

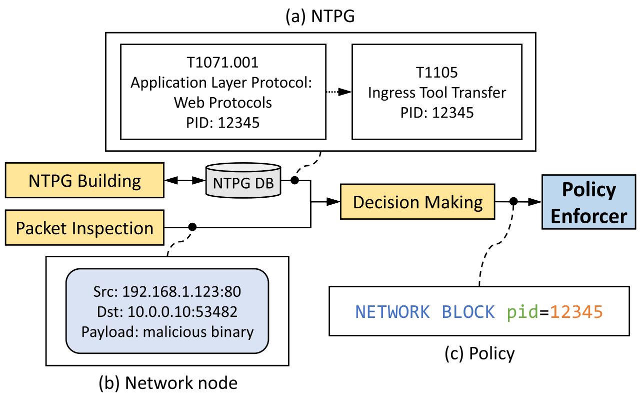


FIGURE 6. The generation of the policy based on constructed NTPG and inspected network information.

Classification (CAPEC) [40] database. This database categorizes the severity of each TTP into four levels: Low, Moderate, High, and Critical, with our system assigning numerical values to these categories as delineated in Table 2. The methodology for defining the severity of TTP can be adjusted by operators as required. If the cumulative score of the NTPG, as determined by the CAPEC criteria, exceeds a predefined threshold,[2] the corresponding events are classified as malicious actions. The attack analyzer, based on these scores, delineates policies, enacting packet inspection rules for processes that attain a score yet remain below the threshold, and packet block rules for processes that either surpass this threshold or are identified as malicious via packet inspection. Policies related to processes adjudged as engaging in malicious activities are subsequently transferred to the policy enforcer, facilitating the inhibition of the network activities from such processes.

The attack analyzer formulates packet filtering rules for processes subjected to a final assessment regarding their malicious nature. Each packet filtering rule is defined by the PID to which it applies, coupled with a specified action that governs the process's network behavior. Two distinct actions are employed: *block*, which entirely prohibits network activity for the process, and *inspect*, which routes packets to undergo further inspection. The block action is deployed to inhibit the network activities of processes conclusively identified as malicious. Conversely, the inspect action is reserved for processes not definitively classified as malicious but exhibiting potential malicious behavior, necessitating comprehensive scrutiny of their network activities. This approach facilitates targeted traffic inspection to minimize disruptions to network performance. The decisions rendered by the attack analyzer are transferred to the policy enforcer, which then implements packet filtering rules to restrict the network activities of malicious processes.

Figure 6 illustrates how our system synthesizes the NTPG and incorporates network packet data for decision-making, following the scenario shown in Figure 4. The attack analyzer constructs the NTPG by evaluating the PID associations and the causal linkages between two TTP nodes, subsequently interconnecting them. The resulting NTPG is depicted in Figure 6(a). Concurrently, the packet exchanged between the host and the C&C server is being inspected. Since these packets stem from the host events, they are inherently linked to the corresponding high-level TTP node that represents the host event. In this analysis, the packet inspection module detects malicious code within the packet associated with the binary download event (i.e., Figure 6(b)), relaying this finding to the decision-making logic. As a result, the network packet is classified as malicious, leading to the classification of the behavior exhibited by the process with PID 12345 as malicious as well. This assessment leads to the development of a policy, depicted in Figure 6(c), aimed at halting all network activities originating from the implicated process.

C. POLICY ENFORCEMENT

Given that botnet attacks are orchestrated via the network, curtailing network activity effectively prevents attackers from compromising additional hosts within the network. This is accomplished by the policy enforcer, which implements network rules within the packet processing pipeline to obstruct the network activities of malevolent processes. The policy DB catalogues processes identified as malicious or potentially malicious by the attack analyzer, recording their information and prescribed actions—specifically, whether to *block* or *inspect* their network activities—without the network session details of the process as delineated in section VI-B. This omission is deliberate, as network sessions can be initiated at any moment, necessitating the dynamic acquisition. To dynamically gather network session data generated by a process, the policy enforcer obtains that data (i.e., 5-tuple) alongside the PID from the event handler. Subsequently, packets originating from the processes are intercepted and blocked, contingent upon a comparison with the malicious process information maintained in the policy DB.

Moreover, *BotFence* conducts inline and real-time processing of packets via the *packet processing pipeline*, a hardware accelerator embedded within the SmartNIC designed to offload and manage packet flows. This pipeline facilitates the seamless transit of packets to and from the host, thereby allowing interaction with all traffic that traverses the host. Within this infrastructure, the packet filtering mechanisms are implemented, enabling the application of *network rules* to selectively filter and obstruct packets based on predefined criteria. When a process, identified as malicious through its PID, attempts to initiate a network session, the policy enforcer intervenes by applying a *network rule* specifically tailored to inhibit the network activity of the process. The *block* rule outright prevents network activity for flagged processes, whereas the *inspect* rule directs traffic from processes under suspicion of malicious intent to a packet inspection module for detailed analysis. This strategy ensures

[2]We referred Holmes [7] for calculation of the severity to the numerical value and decision of the threshold.

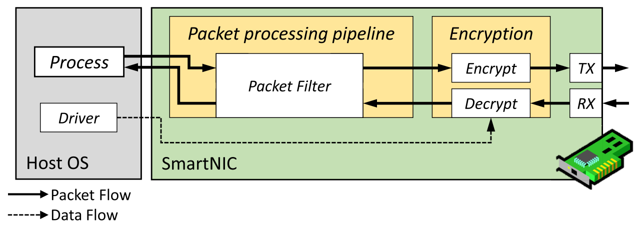
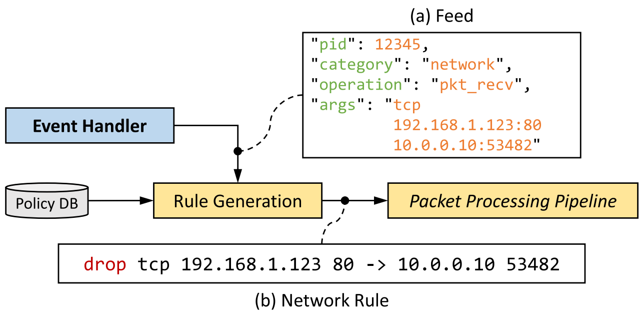


FIGURE 7. ***BotFence* utilizes encryption module to offload packet encryption to the SmartNIC for packet inspection.**



```
"pid": 12345,
"category": "network",
"operation": "pkt_recv",
"args": "tcp
        192.168.1.123:80
        10.0.0.10:53482"
```

```
drop tcp 192.168.1.123 80 -> 10.0.0.10 53482
```

FIGURE 8. The generation of network rule by the network feed of the process which violates deployed policy.

that benign traffic remains uninspected, thereby preserving optimal network performance without the encumbrance of unnecessary scrutiny.

However, adversaries may encrypt packets to obfuscate their activities within network traffic. To handle such encrypted packets, our system leverages the advanced capabilities of SmartNIC's packet encryption offloading technology [41]. Initially, the system employs mechanisms such as kernel TLS (kTLS) offloading, thereby delegating the task of packet encryption to the hardware level. This approach enables the host operating system to transmit packets to our system in an unencrypted state, accompanied by pertinent session information. Figure 7 delineates the packet encryption offloading of *BotFence*. The process commences with the host dispatching encryption session data to our system via its driver. Subsequently, our system scrutinizes the headers of packets destined for external hosts, aligning them with the session information to ensure the encryption of packets prior to their exit from the pipeline. For incoming packets that are encrypted, we employ the stored session and key data to decrypt the packets before their introduction into the pipeline. Thus, the adoption of encryption offloading technology permits the host operating system to relay packets to our system without encryption, thereby enabling our system to examine the packet payloads prior to encryption. Conversely, packets received by the host undergo decryption before entering the pipeline, facilitating the inspection of packet payloads. The inspection logic is strategically integrated within the pipeline, both preceding encryption and following decryption, enhancing its capacity to conduct thorough inspections of packets in their plaintext form.

Figure 8 shows how *BotFence* inhibits the network activities of malicious processes. First of all, as delineated in Figure 6(c), the policy is instituted in our system to obstruct all network interactions of the process PID 12345. Next, the event feed depicted in Figure 8(a) is generated from the host, which means to transfer malicious code to the host (i.e., (3) in Figure 4). Upon detection of the policy violation by the feed, the policy enforcer deploys rule to obstruct the network session associated with PID 12345. Figure 8(b) delineates the enforcement of a network rule designed to halt the download session by intercepting packets routed from 192.168.1.123:80 to 10.0.0.10:53482. To block future network activities emanating from PID 12345, the policy enforcer continually acquires feeds from the event handler, thereby ensuring the proactive blockade of any network engagement by the specified process.

VII. EVALUATION

A. IMPLEMENTATION

A prototype of *BotFence* was developed leveraging commodity SmartNIC to evaluate the practicality and performance efficiency. The development of each module was facilitated with NVIDIA BlueField-2 [17] through the utilization of the DOCA API [42]. The host agent employs the `libbpf` [43] library, enabling an eBPF [44] program to trace kernel probes, thereby gathering essential events occurring within the host. These events are subsequently conveyed to the SmartNIC utilizing the DMA channel. Within our system, the ARM core in the SmartNIC is tasked with event collection, NTPG generation, decision-making processes, policy formulation, and rule establishment. The system's generated NTPGs and policies are preserved within an in-memory database, a strategic choice to expedite query responses. We also utilizes the in-built packet processing pipeline from the SmartNIC for enhanced packet processing speed. This pipeline facilitates the processing of packets in accordance with the predefined rules. Moreover, the DPI module plays a pivotal role in conducting thorough packet inspections. This holistic approach underscores the prototype's capacity to not only validate the proposed system's feasibility but also its effectiveness in optimizing network security measures through advanced hardware integration and software engineering practices.

B. SECURITY EVALUATION

Figure 9 depicts a simulated botnet infection attack scenario, utilized to illustrate the detection and mitigation of attacks by *BotFence*. Such attacks frequently commence due to the inadvertent installation of malware by users [45], [46]. Initially, the malware on the host establishes encrypted communication with the Command and Control (C&C) server, aiming to download and install a rootkit. This is achieved by (1) dispatching a "hello" packet to the C&C server's IP address, obtained via DNS lookup, and employing the HTTPS protocol to masquerade the traffic as legitimate

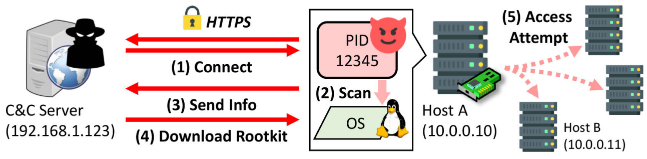


FIGURE 9. **A security evaluation scenario consisting of 5 stages. The infiltrated malware in the Host A and attempted to establish communication with the C&C server. To obfuscate its actions, the attacker leveraged an encrypted session.**

```
{
"pid": 12345,
"category": "process",
"operation": “clone",
"args": "uname -a"
}
```

```
{
"pid": 12345,
"category": "file",
"operation": "read",
"args": "/etc/passwd"
}
```

```
{
"pid": 12345,
"category": "file",
"operation": "read",
"args": "/etc/cpuinfo"
}
```

(a) Feeds of Stage (2)

```
{
  "pid": 12345,
  "category": "network",
  "operation": "pkt_send",
  "args": "192.168.1.123:443"
}
```

(b) Feeds of Stage (3)

FIGURE 10. **The host events of the attack scenario (Figure 9) are transformed into the feeds. (a) corresponds to Stage (2), while (b) corresponds to Stage (3). The host events of other stages are formulated in a similar manner.**

```
IF EXISTS feed WHERE category='process'
AND operation='clone' AND INCLUDE args.cmd='uname -a'
THEN GENERATE(pid, 'T1082')
-------------------------------------------------------
IF EXISTS feed WHERE category='file' AND operation='read'
AND args.filename= '/etc/passwd' THEN GENERATE(pid, 'T1082')
-------------------------------------------------------
IF EXISTS feed WHERE category='file' AND operation='read'
AND args.filename='/proc/cpuinfo' THEN GENERATE(pid, 'T1082')
```

(a) TTP generation rule of Stage (2)

```
IF EXISTS feed WHERE category='network'
AND operation='pkt_send' AND args.dst_port=443
AND EXISTS ntpg WHERE id=pid AND INCLUDE ttp='T1082'
THEN GENERATE(pid, 'T1041')
```

(b) TTP generation rule of Stage (3)

FIGURE 11. **TTP generation rule correlates the feed into specific TTP. For enhanced clarity, the rules are articulated utilizing a SQL-like syntax.**

web activity with encryption. Upon successful connection to the C&C server, (2) the malware proceeds to gather internal system data, such as kernel version of operating system, user account list and cpu information. This information is (3) promptly relayed to the C&C server, enabling the attacker to tailor a compatible rootkit. Subsequently, (4) the attacker transmits the customised rootkit back to the host via an established network session, initiating its installation. The harvested system information and the deployed rootkit serve dual purposes: they facilitate further attacks and solidify the attacker's presence within the host system, (5) thereby transforming the infected host into a strategic vantage point for launching additional attacks on other hosts within the network.

Figure 10 and Figure 11 show the feed and TTP generation rule for Stages (2) and (3) of the attack scenario

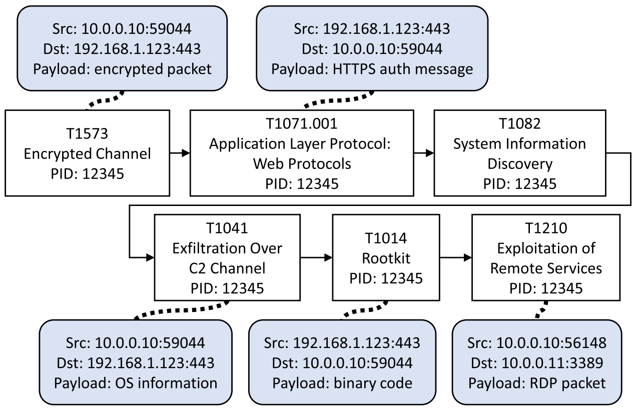
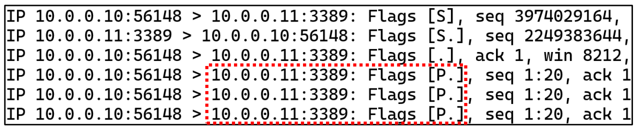


FIGURE 12. **Network enhanced TTP Provenance Graph (NTPG) related to the attack scenario (Figure 9).**

respectively. During Stage (2), the malware attempts to acquire system information, as feed depicted in Figure 10(a). This event is subsequently mapped to the TTP through the generation rule (i.e., Figure 11(a)), which classifies the execution of the `uname -a` command and the reading of the `/etc/passwd` and `/proc/cpuinfo` files as *System Information Discovery* (T1082) [47]. In Stage (3), the malware endeavors to transmit the data regarding the operating system to the C&C server for subsequent exploitation. This event is refined in the feed as depicted in Figure 10(b) and is associated with the TTP through a generation rule (i.e., Figure 11(b)). This correlation identifies the network activity of the process that is already associated with T1082 in the NTPG. Consequently, the event of Stage (3) is characterized as *Exfiltration Over Command and Control Channel* (T1041) [48] and incorporated into the NTPG. The other stages, (1), (4), and (5), are analogously structured into the TTPs in this way as well.

Figure 12 represents the NTPG that encapsulates the attack scenario through the amalgamation of the host and network events as discerned by *BotFence*. The captured host event sequences represented in the graphs, akin to those found in prior research like RapSheet [5], does not integrate network packet data, a gap *BotFence* addresses. Our system extends the analysis to encompass extensive network packet data generated during attack scenarios, integrating this data with the host events for enhanced detection accuracy. Moreover, despite the communication is encrypted, we encounter no challenges in inspecting the packets due to the implementation of encryption offloading. For the clear presentation, the payload item of network information node (the blue boxes in Figure 12), which normally represent specific strings identified within malicious packet payloads, is simplified.

The NTPG is structured into six distinct stages shown in Figure 12, mirroring the phases delineated in the attack scenario. The initial TTP involves malware sending an encrypted *hello* packet to the C&C server for authentication and session establishment (Stage (1)), is categorized under *Encypted*



```
IP 10.0.0.10:56148 > 10.0.0.11:3389: Flags [S], seq 3974029164,
IP 10.0.0.11:3389 > 10.0.0.10:56148: Flags [S.], seq 2249383644,
IP 10.0.0.10:56148 > 10.0.0.11:3389: Flags [.], ack 1, win 8212,
IP 10.0.0.10:56148 > 10.0.0.11:3389: Flags [P.], seq 1:20, ack 1
IP 10.0.0.10:56148 > 10.0.0.11:3389: Flags [P.], seq 1:20, ack 1
IP 10.0.0.10:56148 > 10.0.0.11:3389: Flags [P.], seq 1:20, ack 1
```

FIGURE 13. The result of `tcpdump` log from the Host A. The attacker endeavors to gain access to another host (Host B), but this attempt is thwarted by a network rule implemented by *BotFence*.

Channel (T1573) [49] and *Application Layer Protocol: Web Protocols* (T1071.001) [39]. Analysis of the network nodes reveals the transmission of an encrypted authentication packet from the C&C server (192.168.1.123) to the internal Host A (10.0.0.10) via the HTTPS default port 443, enabling *BotFence* to identify and potentially thwart attack preparations at the C&C access point. The next TTP focuses on the collection of internal system information (Stage (2)), termed *System Information Discovery* (T1082) [47]. This phase is internal to the host, with no corresponding network events, hence the absence of a network node. The fourth TTP, characterized by the exfiltration of collected data through the C&C access channel (Stage (3)), is labeled *Exfiltration Over C2 Channel* (T1041) [48]. Here, a network event is identified, showcasing the transmission of OS information from Host A to the C&C server.

The final two TTPs in Figure 12 provide critical insights into the attack scenario, indicating the completion of establishing a foothold within the host and the subsequent initiation of a genuine threat to the network. The fifth TTP involves the delivery of a rootkit to the host (Stage (4)), leveraging the previously leaked internal information, and is designated as *Rootkit* (T1014) [50]. One network event is again observed, indicating the transfer of the binary code (i.e., rootkit from the C&C server to Host A) highlighting the holistic approach of *BotFence* in detecting and analyzing botnet malware activities. Finally, the last TTP, which indicates the attempt to access an additional host via an installed rootkit (Stage (5)), is represented as *Exploitation of Remote Services* (T1210) [51].

The analysis conducted via the NTPG reveals that the rootkit has been downloaded via the malware process, which subsequently utilizes the rootkit for accessing additional internal hosts. Consequently, it necessitates the creation and deployment of a network rule specifically designed to inhibit the network activities associated with the process. Figure 13 shows the network session aimed at accessing another host, denoted as Host B (10.0.0.11), through the Remote Desktop Protocol (RDP) being effectively neutralized by *BotFence*, as corroborated by the `tcpdump` log (i.e., a red dotted box). The session initiation between Host A and Host B is observed; however, upon detection of RDP utilization for communication, our system categorizes this as an endeavor for lateral movement and subsequently obstructs the session employing a network rule to drop the packets. Consequently, despite the attacker's initial successful infiltration, their efforts to proliferate the attack to additional hosts are impeded by *BotFence*, thereby rendering any further packet exchange unachievable.

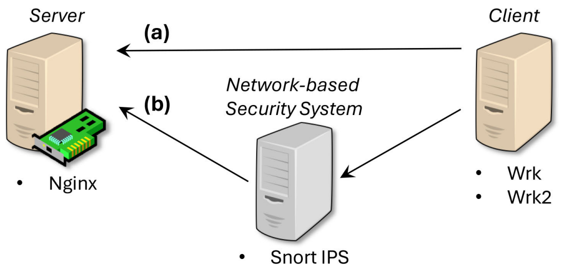


FIGURE 14. Test environments for performance evaluation; *BotFence* is deployed on the server machine. The route (a) is configured for *BotFence*, and the route (b) is for Snort IPS.

The efficacy of *BotFence* in mitigating the attack scenario can be attributed to its advanced detection of suspicious activities via the NTPG, which facilitates the visualization of host event sequences and the analysis of network packet payload. Leveraging encryption offloading capabilities within our system, it became possible to discern that a binary—critical evidence indicative of attack behavior—had been retrieved from the network packets. Furthermore, the execution of the downloaded rootkit binary and the subsequent attempts to compromise remote services on other hosts were identifiable through NTPG. This comprehensive detection mechanism enabled our system to successfully obstruct the attack.

C. PERFORMANCE EVALUATION

BotFence aggregates events from the host, conducts analyses, and integrates these findings with outcomes from network packet scrutiny to ascertain the presence of malicious activities. Consequently, potential impacts on system and host performance due to the operational overhead of our system can be categorized into two primary areas: monitoring of host events and monitoring of network packets. The overhead associated with monitoring host events arises during the event processing and malicious activity determination phases. By measuring the duration required for our system to collect, analyze events, and formulate rules for malicious activity, the extent of this overhead can be assessed. Similarly, the overhead for monitoring network packets is encountered during the packet inspection phase. Assessing the impact on network performance involves comparing network throughput and latency under normal conditions with those observed when our system is operational, utilizing the SmartNIC capabilities.

For the evaluation, an experimental setup comprising two machines—a server and a client, both equipped with Intel Xeon Silver CPUs and 64GB of RAM—was employed as Figure 14. The server was outfitted with an NVIDIA BlueField-2 [17], whereas the client utilized a standard 40G Ethernet NIC, with both machines interconnected via a 40G Ethernet link. *BotFence* was installed on the server, where it functioned to detect and assess attack activities targeting the server. We conducted network overhead measurements

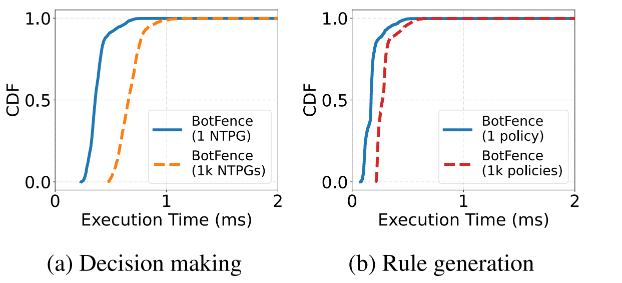


FIGURE 15. (a) Decision making time comparison with the increasing number of NTPGs. (b) Rule generation time comparison with the increasing number of policies.

for the L7 HTTP protocol by deploying the Nginx [52] web server on the server machine. Requests were generated using the HTTP performance measurement tools wrk [53] and wrk2 [54] on the client machine.

1) REAL-TIME EXECUTION

Contrary to conventional systems that accumulate host events for subsequent analysis, *BotFence* is designed to identify attacks in real-time and establish policies to inhibit the network activities of malicious processes. To substantiate this capability, an assessment of the execution latency intrinsic to the operational flow of our system is conducted, establishing the viability of real-time execution. The methodology employed by our system for the identification of attacks and the subsequent enactment of defences can be succinctly categorized into two distinct stages: decision-making and policy enforcement. The decision-making encapsulates the sequence from event collecting, through the NTPG generation and network packet inspection, to the formulation of policy. Conversely, the policy enforcement encompasses the derivation of network rules from the monitored events for the application of the formulated policy. Consequently, the execution time of the decision-making phase is contingent upon the quantity of the NTPGs, while the policy enforcement phase is influenced by the number of policies. Through measurement of the execution durations associated with these phases, an empirical understanding of the temporal efficiency with which our system operates is achieved.

Figure 15a illustrates the temporal metrics associated with the decision-making phase, revealing that the execution time remains relatively stable despite an escalation in the number of NTPGs. Similarly, Figure 15b quantifies the duration required during the policy enforcement phase, echoing the pattern observed in Figure 15a where the execution time exhibits minimal increase with the augmentation of policy count. Notably, it is observed that in excess of 99% of operations within both phases were concluded within a 1ms threshold. This efficiency is a direct consequence of the methodology employed by *BotFence*, which involves the real-time scoring of NTPGs upon the identification of any TTPs, thereby facilitating swift attack detection. Moreover, the strategy of utilizing an in-memory database for the conservation of NTPGs and policies substantially curtails

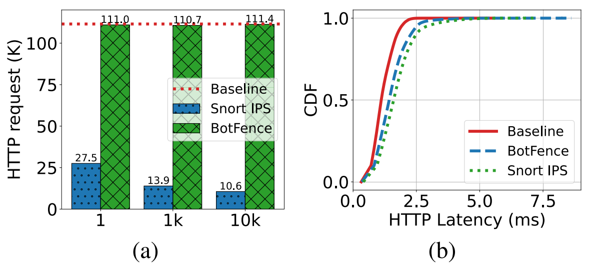


FIGURE 16. (a) L7 throughput comparison with the increasing number of network rules. (b) L7 latency comparison.

both the volume and temporal extent of database queries. This streamlined architecture ensures the capability of our system to perform attack detection and defense implementation in a real-time.

2) NETWORK OVERHEAD

BotFence scrutinizes network packets, specifically focusing on the Layer 7 (L7) content to determine the presence of malicious payloads. Operating inline with the packet processing pipeline, packet inspection could potentially introduce a notable overhead to the host's networking capabilities. This experiment aimed to evaluate the impact on throughput and latency for L7 packet inspection to quantify the overhead induced by our system. To assess the efficacy of our sytsem in packet inspection, we conducted a comparative analysis with Snort [55], a widely recognized software-based security system that employs DPI. Snort is capable of operating in both passive and inline modes for scrutinizing packets. Given that our system is designed to inspect packets concurrently within the packet processing pipeline, we specifically focused on comparing its performance against the inline mode of Snort. According to the findings illustrated in figure 16, the throughput for L7 packets under inspection of our system demonstrates parity with the baseline network performance, indicating no discernible degradation. Further analysis, varying the number of network rules, revealed a consistent network performance, unaffected by the rule quantity. In contrast, Snort demonstrates the throughput that is less than 25% compared to our system, with its performance further diminishing as the rule count increases. Figure 16b presents the latency measurements for L7 packets, showing a marginal increase in latency compared to the baseline, yet outperforming Snort. These outcomes are attributed to the integration of the specialized DPI engine designed for high-speed packet analysis. The engine, being hardware-based, facilitates rapid packet inspection, enabling our sytem to conduct thorough attack detection without compromising network performance, even as the volume of network rules increases.

VIII. DISCUSSION

A. LIMITATION OF SmartNIC RESOURCES

SmartNICs deliver foundational network functionalities in addition to advanced packet processing capabilities. Despite this, such devices possess comparatively limited computational resources compared to conventional hosts. For instance, the Bluefield-2 SmartNIC, employed within our framework, is equipped with an ARM processor and 16GB of memory. While this configuration suffices for the processing of network packets, it may exhibit constraints when tasked with more intricate operations, such as the analysis of host events. The direct execution of these complex tasks on SmartNICs may precipitate performance detriments.

Nonetheless, we have substantiated the feasibility of real-time detection via the deployment of *BotFence*, as delineated in Section VII-C1. This affirmation underscores the potential of SmartNICs to maintain real-time operational efficiency, notwithstanding their limited resource. Our methodology with SmartNICs prioritizes the maximization of packet monitoring acceleration. To this end, we leverage the DPI engine to conduct packet analyses without compromising network throughput. By amalgamating traffic analysis and security surveillance functionalities within SmartNICs, our system effectively diminishes the CPU burden on hosts and augments the overall system efficiency.

Despite the relatively high cost and limited resources associated with SmartNIC, its advantages in enhancing network performance are substantial. SmartNIC offloads network packet processing from the server's main resources, thereby significantly boosting network efficiency. Additionally, its isolated processing environment, separate from the host operating system, fortifies network security by enabling security functions at the hardware level. Consequently, SmartNICs are extensively employed in high-performance server environments, including data centers, cloud service providers, and large-scale enterprise networks. By leveraging the capabilities of SmartNIC, our system can be effectively deployed in networks that demand both high security and superior performance.

B. ADOPTION OF DECISION MAKING METHOD

BotFence employs a dual-faceted approach to botnet detection, integrating host event analysis with network packet inspection to identify security threats. This combined strategy enables our system to detect anomalies by scrutinizing host events while concurrently monitoring network traffic for suspicious packet transmissions. For host event analysis, our system utilizes the Holmes methodology [7], which heavily relies on the CAPEC database. This repository provides a comprehensive range of attack patterns and vulnerabilities, facilitating precise identification and categorization of security threats. Although this rule-based method offers high-speed detection, it is less effective in identifying new attacks. The accuracy of host event analysis and the detection of novel attacks can be enhanced by incorporating insights from existing anomaly-based detection studies [9], [10], [11]. However, employing advanced analytical techniques like graph analysis may present challenges for real-time execution. Our system is designed to separate host event analysis and packet inspection processes, allowing for asynchronous operation. This separation optimizes the use of each analytical method without compromising real-time performance efficiency.

C. DETECTION OF PROLONGED ATTACK

In the context of malware-based automated attacks, attackers often employ an incubation period to obscure the signs and processes of their actions. Botnet infection attacks similarly rely on malware-based automation, enabling them to conceal their activities, such as accessing C&C servers after a prolonged incubation period. When attacks are executed over extended durations, traditional detection methods struggle to identify them due to issues like log loss. This challenge persists in data provenance graph-based attack detection, prompting the development of various solutions. For instance, Holmes [7] and Rapsheet [5] transformed the raw provenance graph into an alert graph, introducing the concept of TTP to represent attack sequences. Our system adopts a similar approach by introducing the NTPG. Borrowing from Rapsheet's methodology, our system does not retain the entire provenance graph but instead organizes and stores data identified as actual attacks in the form of NTPG. This approach minimizes the data size required for attack detection, enhancing data preservation and enabling the detection of attacks that unfold over extended periods.

D. REAL-WORLD DEPLOYMENT OF SYSTEM

Our security evaluation experiments were conducted not in a real-world environment, but through simulations designed to closely replicate attack scenarios. Specifically, we emulated an automated botnet expansion attack, adhering as closely as possible to the lifecycle of an actual botnet as detailed in Section II-A. The simulated attack process involved the installation of malware on the host system, and subsequent communication with a C&C server to receive and install a customized rootkit for the system environment.

During these simulated attack scenarios, our system collected host event information and integrated it with network packet data to generate a NTPG. The security evaluation demonstrated the NTPG creation process, showing the feed generated during event collection and the generation rules to configure TTP. This integration helps network security operators analyze attacks by visualizing the attack process through the NTPG, which is linked to the collection of network packets, providing crucial insights into attack methodologies. Through these processes, our system effectively detects attacks and provides comprehensive information on the nature of attacks, thereby enhancing our readiness for potential security incidents.

IX. CONCLUSION

In light of the escalating magnitude and complexity of botnet attacks, we introduces a novel system, *BotFence*, which amalgamates host event and network packet payload inspection to optimize the detection and mitigation of botnet infections. *BotFence* leverages SmartNIC to offload analysis and construct NTPG for detailed investigation while concurrently preventing the diminution of network performance. Through the execution of simulated attack scenarios, we demonstrated that our system possesses the capability to efficaciously identify botnet activities. Furthermore, these simulations substantiated the proficiency of *BotFence* in obstructing attack maneuvers in real-time, affirming its efficacy without compromising network performance.

REFERENCES

[1] (2023). *The Global Risks Report 2023 18th Edition*. [Online]. Available: https://www3.weforum.org/docs/WEF_Global_Risks_Report_2023.pdf

[2] (2024). *Botnet and IoT Security Trends 2024*. [Online]. Available: https://www.ustelecom.org/wp-content/uploads/2024/03/USTelecom-Botnet-and-Security-Trends-2024.pdf

[3] A. Arfeen, S. Ahmed, M. A. Khan, and S. F. A. Jafri, "Endpoint detection & response: A malware identification solution," in *Proc. Int. Conf. Cyber Warfare Secur. (ICCWS)*, Nov. 2021, pp. 1–8.

[4] S. Bhatt, P. K. Manadhata, and L. Zomlot, "The operational role of security information and event management systems," *IEEE Secur. Privacy*, vol. 12, no. 5, pp. 35–41, Sep. 2014.

[5] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 1172–1189.

[6] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. Stoller, and V. Venkatakrishnan, "SLEUTH: Real-time attack scenario reconstruction from COTS audit data," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 487–504.

[7] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. N. Venkatakrishnan, "HOLMES: Real-time APT detection through correlation of suspicious information flows," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 1137–1152.

[8] M. N. Hossain, S. Sheikhi, and R. Sekar, "Combating dependence explosion in forensic analysis using alternative tag propagation semantics," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 1139–1155.

[9] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "UNICORN: Runtime provenance-based detector for advanced persistent threats," 2020, *arXiv:2001.01525*.

[10] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. A. Gunter, and H. Chen, "You are what you do: Hunting stealthy malware via data provenance analysis," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–24.

[11] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. N. Venkatakrishnan, "POIROT: Aligning attack behavior with kernel audit records for cyber threat hunting," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1795–1812.

[12] F. Dong, S. Li, P. Jiang, D. Li, H. Wang, L. Huang, X. Xiao, J. Chen, X. Luo, Y. Guo, and X. Chen, "Are we there yet? An industrial viewpoint on provenance-based endpoint detection and response tools," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2023, pp. 2396–2410.

[13] X. Wang, K. Zheng, X. Niu, B. Wu, and C. Wu, "Detection of command and control in advanced persistent threat based on independent access," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.

[14] Z. Zhao, H. Sadok, N. Atre, J. C. Hoe, V. Sekar, and J. Sherry, "Achieving 100Gbps intrusion prevention on a single server," in *Proc. 14th USENIX Symp. Operating Syst. Design Implement.*, 2020, pp. 1083–1100.

[15] R. Brewer, "Advanced persistent threats: Minimising the damage," *Netw. Secur.*, vol. 2014, no. 4, pp. 5–9, Apr. 2014.

[16] M. Marchetti, F. Pierazzi, M. Colajanni, and A. Guido, "Analysis of high volumes of network traffic for advanced persistent threat detection," *Comput. Netw.*, vol. 109, pp. 127–141, Nov. 2016.

[17] (2023). *Nvidia Bluefield-2 DPU*. [Online]. Available: https://www.nvidia.com/en-us/networking/products/data-processing-unit/

[18] P. N. Bahrami, A. Dehghantanha, T. Dargahi, R. M. Parizi, K.-K. R. Choo, and H. H. Javadi, "Cyber kill chain-based taxonomy of advanced persistent threat actors: Analogy of tactics, techniques, and procedures," *J. Inf. Process. Syst.*, vol. 15, no. 4, pp. 865–889, 2019.

[19] M. Antonakakis et al., "Understanding the Mirai botnet," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 1093–1110.

[20] S. R. Team, "Emotet exposed: Looking inside highly destructive malware," *Netw. Secur.*, vol. 2019, no. 6, pp. 6–11, Jun. 2019.

[21] S. Shin, G. Gu, N. Reddy, and C. P. Lee, "A large-scale empirical study of conficker," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 2, pp. 676–690, Apr. 2012.

[22] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang, "On the analysis of the Zeus botnet crimeware toolkit," in *Proc. 8th Int. Conf. Privacy, Secur. Trust*, Aug. 2010, pp. 31–38.

[23] (2023). *Necurs: Uncovering the Sophisticated Botnet*. [Online]. Available: https://www.blackhatethicalhacking.com/articles/necurs-uncovering-the-sophisticated-botnet/

[24] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, "Measuring and detecting fast-flux service networks," in *Proc. Ndss*, 2008, pp. 1–12.

[25] U. J. Braun, A. Shinnar, and M. I. Seltzer, "Securing provenance," in *Proc. 3rd USENIX Workshop Hot Topics Secur.*, 2008, pp. 1–20.

[26] S. Gaonkar, N. F. Dessai, J. Costa, A. Borkar, S. Aswale, and P. Shetgaonkar, "A survey on botnet detection techniques," in *Proc. Int. Conf. Emerg. Trends Inf. Technol. Eng.*, Feb. 2020, pp. 1–6.

[27] Y. Xing, H. Shu, H. Zhao, D. Li, and L. Guo, "Survey on botnet detection techniques: Classification, methods, and evaluation," *Math. Problems Eng.*, vol. 2021, pp. 1–24, Apr. 2021.

[28] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *J. Netw. Comput. Appl.*, vol. 36, no. 1, pp. 16–24, 2013.

[29] X. Zhang, C. Li, and W. Zheng, "Intrusion prevention system design," in *Proc. 4th Int. Conf. Computer Inf. Technol.*, 2013, pp. 375–382.

[30] N. Chakraborty, "Intrusion detection system and intrusion prevention system: A comparative study," *Int. J. Comput. Bus. Res. (IJCBR)*, vol. 4, no. 2, pp. 1–8, 2013.

[31] T. Kovanen, G. David, and T. Hämäläinen, "Survey: Intrusion detection systems in encrypted traffic," in *Proc. 16th Int. Conf.*, 2016, pp. 281–293.

[32] K. Özgun, A. Tosun, and M. Sandıkkaya, "A recommender system to detect distributed denial of service attacks with network and transport layer features," in *Proc. 10th Int. Conf. Inf. Syst. Secur. Privacy*, 2024, pp. 390–397.

[33] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita, "Botnet in DDoS attacks: Trends and challenges," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2242–2270, 4th Quart., 2015.

[34] T. A. Tuan, H. V. Long, L. H. Son, R. Kumar, I. Priyadarshini, and N. T. K. Son, "Performance evaluation of botnet DDoS attack detection using machine learning," *Evol. Intell.*, vol. 13, no. 2, pp. 283–294, Jun. 2020.

[35] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson, "Analyzing forged SSL certificates in the wild," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 83–97.

[36] (2023). *Kernel TLS Offload—The Linux Kernel Documentation*. [Online]. Available: https://docs.kernel.org/networking/tls-offload.html

[37] (2023). *MITRE ATT&CK*. [Online]. Available: https://attack.mitre.org/

[38] (2023). *Data From Local System, Technique T1005—Enterprise | MITRE ATT&CK*. [Online]. Available: https://attack.mitre.org/techniques/T1005/

[39] (2023). *Application Layer Protocol: Web Protocols, Sub-technique T1071.001—Enterprise | MITRE ATT&CK*. [Online]. Available: https://attack.mitre.org/techniques/T1071/001/

[40] (2023). *Common Attack Pattern Enumerations and Classifications*. [Online]. Available: https://capec.mitre.org/

[41] B. Pismenny, I. Lesokhin, L. Liss, and H. Eran, "TLS offload to network devices," in *Proc. Tech. Conf. Linux Netw.*, 2016, pp. 1–24.

[42] (2023). *DOCA SDK Documentation*. [Online]. Available: https://docs.nvidia.com/doca/sdk/index.html

[43] (2023). *Libbpf*. [Online]. Available: https://github.com/libbpf/libbpf

[44] (2023). *EBPF*. [Online]. Available: https://ebpf.io/

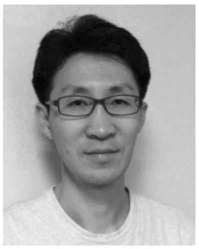
[45] (2021). *Apt1: Exposing One of China's Cyber Espionage Units*. [Online]. Available: https://www.mandiant.com/sites/default/files/2021-09/mandiant-apt1-report.pdf

[46] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang, "A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1851–1877, 2nd Quart., 2019.

[47] (2023). *System Information Discovery, Technique T1082—Enterprise | MITRE ATT&CK*. [Online]. Available: https://attack.mitre.org/techniques/T1082/

[48] (2023). *Exfiltration Over C2 Channel, Technique T1041—Enterprise | MITRE ATT&CK*. [Online]. Available: https://attack.mitre.org/techniques/T1041/
[49] (2023). *Encrypted Channel, Technique T1573—Enterprise | MITRE ATT&CK*. [Online]. Available: https://attack.mitre.org/techniques/T1573/
[50] (2023). *Rootkit, Technique T1014—Enterprise | MITRE ATT&CK*. [Online]. Available: https://attack.mitre.org/techniques/T1014/
[51] (2023). *Exploitation of Remote Services, Technique T1210—Enterprise | MITRE ATT&CK*. [Online]. Available: https://attack.mitre.org/techniques/T1210/
[52] *NGINX: Advanced Load Balancer, Web Server, & Reverse Proxy*. Accessed: Aug. 20, 2024. [Online]. Available: https://www.nginx.com/
[53] (2021). *WRK—A HTTP Benchmarking Tool*. [Online]. Available: https://github.com/wg/wrk
[54] (2019). *WRK2*. [Online]. Available: https://github.com/giltene/wrk2
[55] (2023). *Snort—Network Intrusion Detection & Prevention System*. [Online]. Available: https://www.snort.org/

HYUNMIN SEO received the B.S. and M.S. degrees in electrical engineering from KAIST, where he is currently pursuing the Ph.D. degree with the School of Electrical Engineering. His research interests include programmable network data planes, cyberattack, and cloud security.



SEUNGWON SHIN (Member, IEEE) received the B.S. and M.S. degrees in electrical and computer engineering from KAIST and the Ph.D. degree in computer engineering from the Electrical and Computer Engineering Department, Texas A&M University. He is currently an Associate Professor with the School of Electrical Engineering, KAIST. His research interests include software-defined networking security, dark web analysis, and cyber threat intelligence.

SEUNGSOO LEE received the B.S. degree in computer science from Soongsil University and the M.S. and Ph.D. degrees in information security from KAIST, in 2020. He is currently an Assistant Professor with the Department of Computer Science and Engineering, Incheon National University. His research interest includes developing secure and robust cloud/network systems against potential threats.