DDoSMiner: An Automated Framework for DDoS Attack Characterization and Vulnerability Mining

Xi Ling[1], Jiongchi Yu[2], Ziming Zhao[1], Zhihao Zhou[1], Haitao Xu[1], Binbin Chen[3], and Fan Zhang[1,4](✉)

[1] College of Computer Science and Technology, Zhejiang University, Hangzhou, China
[2] School of Computing and Information Systems, Singapore Management University, Singapore, Singapore
[3] Information Systems Technology and Design, Singapore University of Technology and Design, Singapore, Singapore
[4] Zhengzhou Xinda Institute of Advanced Technology, Zhengzhou, China
fanzhang@zju.edu.cn

Abstract. With the proliferation of Internet development, Distributed Denial of Service (DDoS) attacks are on the rise. As rule-based traffic analysis frameworks and Deep Packet Inspection (DPI) defense measures can effectively thwart many DDoS attacks, attackers keep exploring various attack surfaces and traffic amplification strategies to nullify the defense. In this paper, we propose DDoSMiner, an automated framework for DDoS attack characterization and vulnerability mining. DDoSMiner analyzes system call patterns of the TCP-based DDoS attack family, then generates Attack Call Flow Graph (ACFG) by discerning the differences between DDoS attack traffic and benign traffic. Furthermore, DDoSMiner identifies and extracts drop nodes and pivotal TCP states from the distinctive characteristics of attack traffic, then passes to the symbolic execution framework for exploring variants of the DDoS attack. We collectively analyze six types of TCP-based DDoS attacks, construct the corresponding ACFG, and identify a set of attack traffic variants. The attack traffic variants are evaluated on the widely used Network Intrusion Detection System (NIDS) Snort with three popular rule sets. The result shows that DDoSMiner indeed discovers the new DDoS attack trace, and the corresponding attack traffic can bypass all three defense toolkits.

Keywords: TCP-based DDoS attacks · Attack Call Flow Graph · Symbolic execution

1 Introduction

With the evolution of the Internet, the security issues of the Internet have garnered increasing attention. Among the various threats to networks, Distributed

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

Denial of Service (DDoS) attacks are regarded as one of the most serious and commonly employed attack methods in practice [16,35,37,56,61]. For instance, Cloudflare has reported a DDoS attack which is launched by a botnet comprising approximately 11,000 IP addresses, peaking at an alarming 1.4 Tbps of attack traffic [55].

Although there are various detection and defense techniques for DDoS attacks [9,29,59], the main defense methods rely on traffic scrubbing [50], which requires expensive dedicated hardware. The core idea of this approach is redirecting the traffic of the target to the scrubbing centers of the Internet security service providers, where malicious traffic is identified and filtered. To be more effective, traffic scrubbing has evolved from centralized single-point detection to distributed detection solutions [31]. However, these methods still face challenges, specifically in terms of flexibility. On one hand, these detection methods heavily depends on filtering strategies crafted from known attacks, making it vulnerable to zero-day threats. [23]. On the other hand, middle-box-based detection and defense systems rely on hardware devices [13,22] and lack adaptability to various attack scenarios and network configurations. In addition, these methods increase the cost for transmission and storage, and bring more attack surfaces targeting middleware and cloud platforms [1,12].

Fortunately, with the emergence of Software-Defined Networking (SDN) [15] and Network Function Virtualization (NFV) [19] technologies, research on defense systems based on programmable networks has also pointed out new directions for DDoS detection and defense [54]. Bohatei *et al.* [14] is the first to design a flexible and resilient DDoS detection and mitigation system based on SDN. Although, subsequent studies based on this new network paradigm and network devices (e.g., programmable switches and smart NICs) have improved the flexibility and scalability of defense systems [28,52,57], the real-time response speed and performance overhead need to be further improved.

Despite defense research efforts, the development of new DDoS attacks continues. Conversely, more and more diverse strategies have been shown in DDoS attacks [30,34,43]. Firstly, emergent malware [39,45,46], such as Mirai [2], has notably bolstered the potency of DDoS attacks by rapidly commandeering the ever-increasing Internet of Things devices [33], leading to increased peak traffic and diversified attack vectors. Secondly, many vulnerabilities are constantly being exploited in network protocols, especially those based on the TCP protocol. Bock *et al.* [6] show that attackers exploiting vulnerabilities in the TCP protocol for reflection amplification attacks is a potential new attack way, and the amplification effect produced surpasses that of UDP-based attacks. TCP-based DDoS attacks exploit the inherent characteristics of the TCP protocol. Attackers employ a myriad of strategies and techniques to implement these attacks and evolve them to evade detection. Thus, excavating attack patterns and identifying vulnerabilities in existing protocols and systems is imperative for effective detection.

Symbolic execution has been wilde used for is vulnerability exploitation and patching [4,42]. Its prowess in navigating through intricate branch conditions can achieve a deeper path execution. Further, Selective symbolic execution improves it, as it can analyze multiple execution paths of a program and switch modes between symbolic execution and concrete execution. Its flexibility in testing large and complex systems, such as operating system kernels, gives it superior performance in detecting bugs and vulnerabilities in binary-based projects [47]. Therefore, in this work, we adopt selective symbolic execution to discover more variants of TCP-based DDoS Attacks and recognize the different categories of attack methods at the system level, observing the depth of system calls and the behavior of the TCP protocol during the attack.

In this work, we propose an automated framework termed DDoSMiner, which can characterize DDoS attack patterns and explore variants of DDoS attack traffic. Specifically, DDoSMiner would initially record TCP traffic and generate the corresponding Attack Call Flow Graph (ACFG) for further recognition of the DDoS attack patterns. The key nodes of the ACFG are extracted by differentiating between benign and attack traffic for subsequent symbolic execution analysis. For the symbolic execution module, DDoSMiner explores potential attack traces within the TCP protocol based on reachable path termination key states from the ACFG, generating various reachable candidate attack packet sequences. Based on our experiments, we identify a new attack traffic, which is a variant of SYN Flood Attack related to timestamp obfuscation. The evaluation of the results on three popular rule sets of Snort demonstrates the new attack could bypass all defense rule sets, while traditional DDoS attacks cannot.

In summary, the contributions are as follows:

- We propose an automated framework DDoSMiner for characterizing the system control flow behavior of DDoS attacks and exploring new DDoS attacks.
- We gather 6 TCP-based DDoS attacks and adopt DDoSMiner to generate ACFG for analyzers. Furthermore, we discover a new DDoS attack trace and collect corresponding attack traffic.
- Empirical results show that the attack generated by DDoSMiner successfully evades three popular detection rule sets on NIDS.

2 Background and Related Work

In this section, we provide a brief background for TCP-based DDoS attack detection and defense. Subsequently, we summarize existing DDoS mining/exploit schemes. Finally, we introduce symbolic execution technology, which is used to construct DDoSMiner.

2.1 TCP-Based DDoS Attacks

DDoS attacks refer to attackers control devices on the Internet to generate massive malicious or useless packets to disrupt the target network services. Most DDoS attacks are developed based on TCP [32], and typical categories involve bandwidth attacks and resource exhaustion attacks [16]. This is because the characteristics of the TCP protocol could be exploited by adversaries to transport-layer paralyze target systems or services. For example, a series of TCP-based

DDoS attacks lies in exploiting TCP control packets by deceiving the three-way handshake between the source and target servers, exhausting the resources of the target server, eventually resulting in unavailable services. To resist these attacks, defenders typically adopt various strategies, such as IP address-based access control [21,48,49,53], intrusion detection system (IDS) [5,44,51,60,62], and distributed firewalls [3,24], to filter out forged connection requests and mitigate the impact of attacks.

2.2 DDoS Mining/Exploit Schemes

To mine more DDoS attack strategies, existing solutions mainly involve manual schemes and fuzzing-based. The former mainly mines emerging attacks in a manual manner and requires domain-specific expert experience. Rossow *et al.* [38] propose 14 types of reflective DDoS attacks based on features including protocols, payload sizes, and packet transmission frequency. Hong *et al.* [20] propose two attacks against network topologies by finding that most mainstream SDN controllers are vulnerable to network visibility poisoning. These works rely on expert knowledge and cannot be automatically conducted/explored.

The latter mainly leverages fuzzing to discover new DDoS strategies. Among them, AMPFUZZ [25] introduces a protocol-agnostic approach for UDP vulnerability, significantly enhancing the fuzzing performance of AMPFUZZ with UDP awareness. However, this work only covers states while ignoring state transitions, which has a significant impact on TCP implementation (compared with stateless UDP). For instance, TCP-Fuzz [63] proposes a new strategy for generating effective test cases for TCP stacks by considering the dependencies between inputs. TCP-Fuzz only tests TCP stacks in user space and does not check kernel-level TCP stacks, thus it cannot obtain coverage of their branches and branch transitions. StateDiver [58] is based on fuzzing and uses the discrepancy between the two inputs in the protocol stack as feedback to explore abnormal nodes in TCP implementations (on DPI). While there is a lack of relevant feedback for DDoS. Our work is based on the TCP stack at the Linux kernel level, allowing more precise analysis of TCP's state transitions and using symbolic execution to explore the TCP stack at the source code level.

2.3 Exploration TCP Stack with Symbolic Execution

Symbolic execution is a white-box program analysis technique. It explores multiple possible execution paths by adopting symbolic input values instead of concrete input values. This allows the exploration of various paths a program might take under different inputs, achieving good performance on program analysis/vulnerability detection. However, it has some problems regarding the explosion of path state space.

With the development of constraint satisfaction problems and the emergence of more scalable dynamic methods that combine concrete and symbolic execution [8], concolic testing merges symbolic execution with concrete execution, aiming to automatically discover vulnerabilities and errors in programs. Selective symbolic execution [10] further extends concolic execution, enabling program analysis in real software stacks (user programs, libraries, kernels, drivers, etc.) rather than using abstract models of these layers, and directly operating on binary files.

As a leading work, SYMTCP [51] uses symbolic execution technology to construct adversarial packets targeting TCP implementations. These packets are designed to leverage the discrepancies between Deep Packet Inspection (DPI) middleboxes and end hosts, to achieve eluding attacks. However, this requires the manual review of the TCP stack's source code and manual marking of drop points (serve as termination points) in the Linux kernel.

Based on these studies, we intend to utilize symbolic execution to analyze TCP-based DDoS attack patterns. Different from SYMTCP, our pipeline involves establishing and analyzing the ACFG by tracking TCP behavior on a white-box target. By comparing normal traffic with various categories of attack traffic, we identify drop nodes in the attack and generate related constraints for symbolic execution.

3 Threat Model and Problem Definition

In this section, we describe the threat model in the context of DDoS attacks and defense. We will provide a detailed definition of the ACFG in the following sections, which will be used to outline path constraints.

3.1 Threat Model

Consider the DDoS Attack Defense Architecture as illustrated in Fig. 1. Attackers send a large number of bogus TCP connection requests to the target system through infected computers or devices, aiming to exhaust the resources of the target server and thus prevent legitimate users from accessing the target service. The IDS acts as a middlebox to monitor and report the traffic.

Actually, the TCP protocol on the server can be regarded as a discrete state transition process, and the attack on the transmission protocol can be regarded as a process of finite state machine state transitions. The execution of the TCP protocol on the server can be modeled as a TCP finite state machine for program analysis. The attackers aim to change the state of the TCP state machine by sending probing attack packets and altering the response of the server to these probing packets.

We adopt the TCP stack of the Linux kernel for system-level program analysis. Furthermore, to expand attack scenarios, we assume that attackers can spoof addresses, which means they can disguise the source IP address of their attack traffic. This assumption increases the diversity and applicability of the attacks. Through comparative experiments between attack traffic and benign traffic, we ensure that the anomalies observed on the server side after an attack

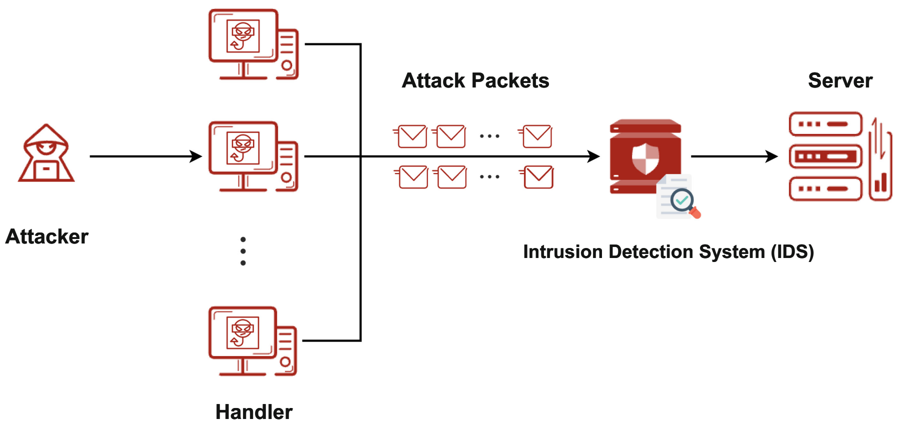

Fig. 1. Illustration of DDoS attack pipeline and IDS.

are due to resource exhaustion attacks caused by the TCP protocol, not bandwidth attacks overwhelmed by high traffic volume, because the server system will not be destroyed by benign packets under the equivalent traffic loads.

3.2 Problem Definition

In this section, we provide a detailed definition of this work and offer a characterization of the ACFG and its elements.

Definition 1: TCP State Machine. Based on the TCP protocol specifications, the Mealy TCP state machine [27,51] can be described as follows,

$$M = (S, I, O, \Sigma, Z)\,, \tag{1}$$

S: The finite non-empty set of states. For instance, the typical set of TCP states includes LISTEN, SYN_RCVD, ESTABLISHED states. $s_0 \in S$ represents the initial state.

I: The input symbol set, representing input events of the state machine, i.e., TCP packets.

O: The output symbol set, represents the output actions of the state machine. For example, sending SYN packets, sending ACK packets, closing connections, etc.

Σ: The state transition function, which defines the transition rules between states, denoted as $S \times I \rightarrow S$. It specifies the next state the state machine will move to, given a certain state and input event.

Z: Output function, which defines the relationship between the output symbol, state, and input event, denoted as $S \times I \rightarrow O$. It specifies which output action the state machine should perform, given a state and input event.

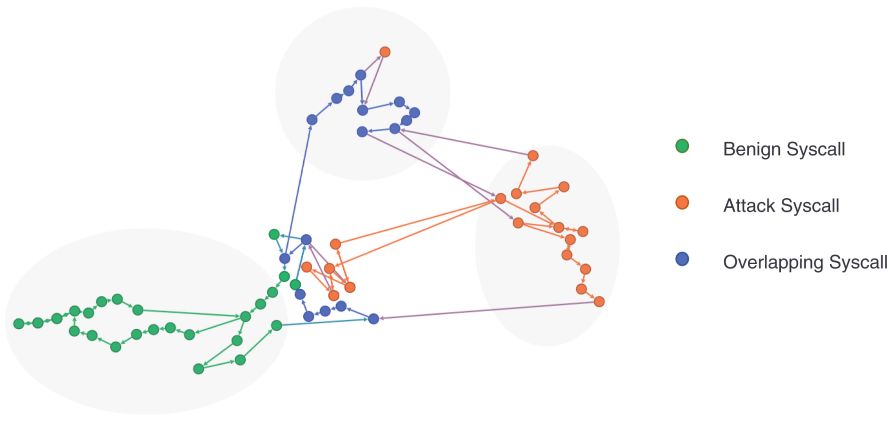

Fig. 2. Syscall interaction analysis based on categories and connectivity.

Definition 2: Attack Call Flow Graph. The ACFG is a directed weighted graph $G = (V, E)$, where each node in the set of vertices represents a function of the TCP stack. The call relationships between functions are represented by directed edges. We extract different classes of ACFGs based on the characteristics of benign traffic and attack traffic.

Definition 3: Malicious Nodes. Malicious nodes represent the function nodes involved in the attack. These nodes are typically the functions abused by attackers in the target system to initiate attacks or bypass security mechanisms. The set of malicious nodes is denoted as V_M.

Definition 4: Critical Nodes. Critical nodes represent important functional nodes in the TCP stack, and their stability and correctness are crucial for the operation of the entire system. Attackers may attempt to destroy the target system through these nodes. After visualizing the system calls of benign and attack traffic, we discover both of them rely on a specific node in a chained call process. These malicious nodes resemble nodes in the normal traffic. While there may be recursive calls within the path, such patterns often accompany the presence of a critical node. Therefore, we further extract the critical nodes from the ACFG.

As shown in Fig. 2, after clustering network nodes based on connectivity and categories, we find that some nodes have an impact on the network's cluster structure. The key to defining critical nodes is to find an optimal subset of nodes in the graph, denoted as $V_C \subseteq V_M$, such that the removal of these nodes has the maximum impact on the network's connectivity [26]. Let the set of critical nodes be denoted as $V_C = \{v_1, v_2, ..., v_l\}$,

$$\lambda_1 = \lambda_{N-1}, \tag{2}$$

$$\lambda_2 = \lambda_{N-l}, \tag{3}$$

$$\lambda_1 \left[-c_0 P_{(0)N-l} + \frac{\lambda_\tau}{2} \sum_{r=1}^{k} c_r P^2_{(r)N-l} \right] \geq \lambda_2 \left[-c_0 P_{(0)N} + \frac{\lambda_\tau}{2} \sum_{r=1}^{k} c_r P^2_{(r)N} \right], \quad (4)$$

where λ_{N-1} represents the eigenvalue of the original matrix, λ_{N-l} represents the eigenvalue of the matrix after removing the set of critical nodes, and λ_τ represents the maximum eigenvalue of the internal coupling matrix between the state variables of each node. The values $c_r > 0, r = 1, ..., k$ represent the connectivity strengths of the r-th sub-network. The matrix $P_{N(r)} = \left[p_{(r)ij}\right]_{N \times N}$ represents the external coupling matrix of the r-th sub-network, used to describe the network topology. The definition of the matrix $P_N = [p_{ij}]_{N \times N}$ is as follows: if there is an edge connecting node i and node j, then $p_{(r)ij} = p_{(r)ji} = -1$, otherwise, it is 0.

The set of critical nodes should also satisfy that the nodes in V_C have the maximum total weighted sum, where $f(v_i)$ represents the attribute value of node v_i,

$$\max \sum f(v_i), v_i \in V_C. \quad (5)$$

According to Definition 2 and Definition 4, we generate the critical nodes as described in Algorithm 1.

Definition 5: Pivotal Nodes. Pivotal nodes represent the distinctive nodes that cause a change of the TCP state. The set of pivotal nodes is denoted as V_P.

Definition 6: Drop Nodes. We focus primarily on nodes that not only have malicious behavior but also play a crucial role in the network. These types of nodes are not just potentially malicious in intent but also have a significant impact due to their critical position in the network structure. In addition, we consider those nodes that represent state transitions in TCP connections because they play an important role in determining the TCP state machine.

Thus, drop nodes are defined as the intersection of malicious nodes and critical nodes union with pivotal nodes, represented as,

$$V_D = V_M \cap V_C \cup V_P. \quad (6)$$

Definition 7: Candidate Attack Sequence. When exploring the TCP state machine M, if the TCP packet $Packet_i \in I$ either reaches our defined termination point or neither causes a change in the TCP state machine's state nor generates any output, then that packet belongs to the candidate attack sequence, as follows,

$$\varSigma(s, Packet_i) = s \wedge Z(s, Packet_i) = \varepsilon. \quad (7)$$

Algorithm 1: Critical Nodes Algorithm

```
Input: Directed graph $G$, node attribute value $f$, eigenvalues of the original
       matrix $\lambda_{N-1}$, maximum eigenvalue of inner coupling matrix $\lambda_\tau$, external
       coupling matrix of $r$th subnetwork $P_{N(r)}$.
Output: $criticalNodes$
1  // Step 1: Initialize criticalNodes and maxImpact
2  $criticalNodes \leftarrow \emptyset$, $maxImpact \leftarrow -\infty$;
3  // Step 2: Traverse each node in the graph G
4  for $node$ in $G.nodes$ do
5      // Step 3: Remove the current node, create graph $G'$
6      $G' \leftarrow G.\texttt{removeNode}(node)$;
7      // Step 4: Initialize the impact of cluster C
8      $C.impact.\texttt{init()}$;
9      // Step 5: Traverse each cluster in graph G
10     for $C$ in $G.clusters$ do
11         // Step 6: Traverse each node in the cluster C
12         for $node$ in $C.nodes$ do
13             // Step 7: The dependency of current and next node
14             $C.conn \leftarrow \mathrm{any}(G.\texttt{hasEdge}(node, node.next) \wedge node \neq node.next)$;
15             $C'.conn \leftarrow \neg\mathrm{all}(G'.\texttt{hasEdge}(node, node.next) \wedge node \neq node.next)$;
16             if $C.conn \wedge C'.conn$ then
17                 // Step 8: Calculate the sum of attribute value and
                   update the impact of cluster C
18                 $\sum f_C \leftarrow \sum(f_{node}$ for $node$ in $C)$;
19                 $C.impact \leftarrow C.impact + P_{N(r)}^2 \cdot (\lambda_{N-1} - \lambda_\tau) \cdot \sum f_{\mathrm{C}}$;
20             end
21         end
22     end
23     if $C.impact > G.initImpact$ and $C.impact > maxImpact$ then
24         // Step 9: Update maxImpact, add node to criticalNode
25         $maxImpact \leftarrow C.impact$;
26         $criticalNodes.\texttt{add}(node)$;
27     end
28 end
```

4 Workflow of DDoSMiner

The overview of the DDoSMiner workflow is illustrated in Fig. 3, which consists of three modules. In Module 1, DDoSMiner traces the kernel for both benign and attack packets, and modeling attacks to generate a visual ACFG. In this stage, we aim to collect all attack paths and critical nodes under different TCP states when attacks occur. Module 2 refers to the symbolic execution phase. The inputs consist of a set of TCP seed packets that drive the selective symbolic execution engine to explore the TCP stack based on the drop nodes. Module 3 is the online verification phase, launching the generated candidate attack sequence to bypass the existing IDS.

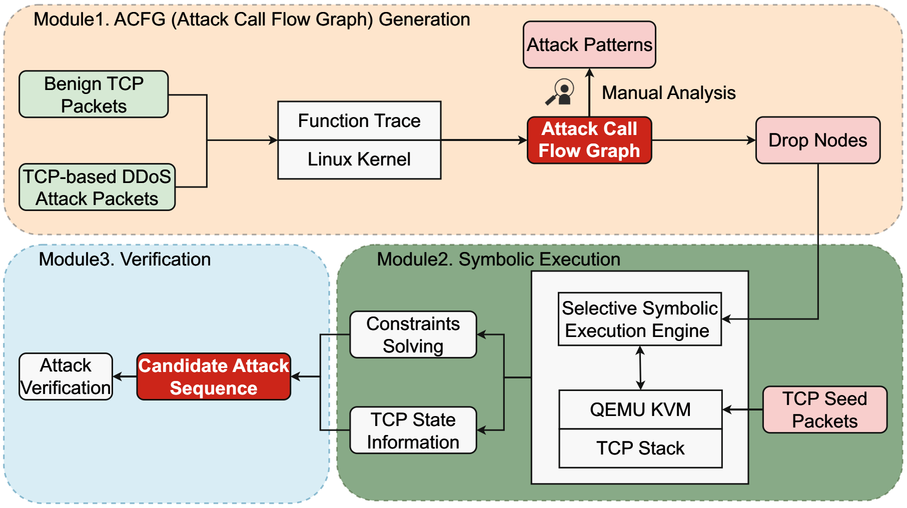

Fig. 3. Workflow of DDoSMiner.

4.1 Generation of Attack Call Flow Graph

The complete TCP state transition involves 11 states [18], and we simplify the states of the TCP finite state machine as shown in Fig. 4. When a TCP connection is established, both the client and server are in the CLOSED state. The server creates a socket and begins listening for incoming remote requests, at which point it enters the LISTEN state. The client initiates a connection by sending a SYN segment (SYN=1) to the server, requesting to establish a connection. Upon receiving the segment, the server responds by sending an ACK and SYN segment (SYN=1, ACK=1) to the client. Meanwhile, the server's state transitions to SYN_RCVD. After receiving the segment, the client sends an ACK to the server. Upon receiving the ACK, the server's state transitions to ESTABLISHED. Then the three-way handshake is completed, and the TCP connection is established.

We establish an experimental environment running on a standard Linux operating system, designed to collect kernel information and recreate TCP-based DDoS attacks from datasets [40,41] for kernel tracing and analysis.

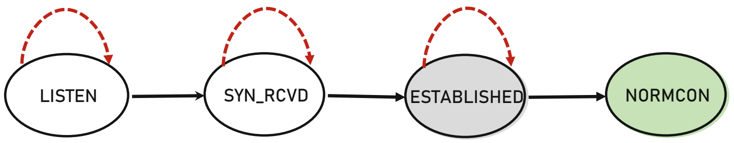

Fig. 4. Simplified TCP states.

Specifically, six categories of attacks are considered, including TCP Connect Flood, TCP SYN Flood, TCP ACK Flood, TCP RST Flood, ACK-PSH Flood, and SYN-RST-ACK Flood attacks. Each attack class abuses TCP connections consumes resources of the target server, or obfuscates network traffic in its unique way. Learning from these categories of attacks allows us to gain a more comprehensive understanding of the DDoS threats based on the TCP protocol and how to effectively identify and respond to these threats.

According to the definitions provided in Sect. 3.2, we generate ACFGs associated with different attack categories and present them through visualization. In Fig. 5, we use different colors to represent different classes of nodes. Blue nodes represent malicious nodes, orange nodes represent critical nodes, and red nodes represent pivotal nodes. The radius of each node reflects the size of its attribute value, with larger nodes indicating higher importance. Directed edges in the graph represent the call relationships between functions, and the edge weights indicate the level of dependence between different functions under the same traffic conditions.

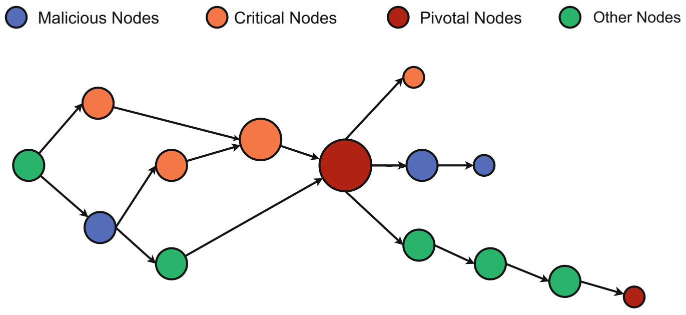

Fig. 5. Diagram depicting paths for benign and attack TCP packets.

4.2 Selective Symbolic Execution

S2E utilizes the symbolic execution engine KLEE [7] and conducts kernel testing through the QEMU simulation system. It also offers an API interface [11] that enables users to customize the scope of symbolic execution, facilitating a seamless transition between symbolic execution and concrete execution modes. Firstly, the symbolic execution engine initiates the running Linux kernel using a TCP socket in the LISTEN state. Subsequently, it provides multiple symbolized TCP packets to the kernel to comprehensively explore the server's TCP stack. The generation of symbolized data packets is divided into two parts:

(i) Generating TCP header packets with various combinations. During the subsequent symbolic execution process, we focus on how changes in TCP header fields such as sequence number, acknowledgment number, data offset, flags,

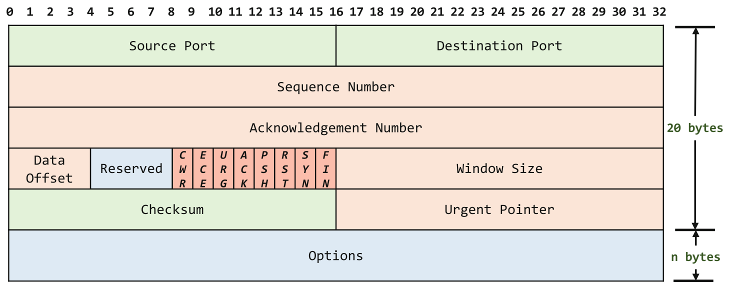

Fig. 6. Symbolized TCP header and options [51].

window size, and urgent pointer affect path exploration. We automatically generate various TCP header segments using script files, combine them with other components, and create TCP seed packets.

(ii) Symbolizing data packets. The input for symbolic execution is not concrete data, it requires symbolizing data values. A TCP header consists of a minimum of 20 bytes of fixed data (as shown in Fig. 6), storing the necessary information for the packet transmission. The 20-byte TCP header does not include options or data. Symbolizing TCP packets is one of the essential tasks in symbolic execution analysis. It allows analysis tools to use symbolic variables instead of concrete data values. This helps simulate various TCP packet transmission paths, identifying potential vulnerabilities or issues.

Consider DDoS attackers typically deliberately mask their source IP addresses and send a large number of false or invalid requests to the target port. Therefore, we do not symbolize the source port and destination port to avoid impacting the performance of symbolic execution. Symbolized fields include sequence number, acknowledgment number, data offset, flags, window size, urgent pointer, and TCP options. Changes in these fields can involve alterations in the TCP state. After constructing the TCP seed packets sent to the Linux host in S2E, we call the symbolization module for processing.

All execution paths for symbolic execution form a tree-like structure known as the symbolic execution tree. We convert the directed weighted graph G corresponding to the ACFG into a directed spanning tree. We utilize a recursive depth-first search (DFS) to explore the graph G and construct a directed spanning tree T, ensuring that no cycles are encountered during the analysis process. This way, we can define symbolic execution path constraints through the generated tree T.

Through symbolic execution, we are able to identify a candidate attack sequence that satisfies the branch termination conditions, and record the path selections and symbolic constraints from the path exploration process in an output document. This enables us to obtain attack patterns on the white-box system, allowing for a detailed analysis of the attacks and the discovery of new attack sequences.

5 Evaluation

In this section, we first introduce the environment and configuration of our experiment. We list the packet variants found by symbolic execution and check whether IDS can prevent the corresponding attack behavior. After that, we in-depth analyze the details of packet variants that can successfully bypass the defense of IDS, while others can not.

5.1 Experiment Configuration

Testbed. We develop the prototype of DDoSMiner based on S2E 2.0 and Linux kernel with S2E extension. The host operating system is Ubuntu 22.04, 64-bit, with a 12-core CPU, specifically the 12-*th* Gen Intel(R) Core(TM) i5-12400, and the GUEST operating system is Debian 11.3, 64-bit, with a 12-core CPU, specifically the 12-*th* Gen Intel(R) Core(TM) i5-12400. Both systems are all running based on Linux kernel v4.9.3. We run S2E in parallel mode with 48 cores, which is the maximum number of processes supported at present.

We use S2E to test the TCP stack implementation in the Linux kernel and switch between the concrete mode and symbolic execution mode. When the program reaches the `tcp_v4_rcv()` code segment, we switch to symbolic execution mode, while the rest of the code segments on the kernel remain in concrete execution mode. Test cases and symbolic constraints are generated when we reach the code segment where our marked drop nodes are located.

IDS and Rule Sets. For evaluation, we use Snort 3 [36] as the deployed IDS by the victim. Given Snort is the foremost open-source NIDS in the world, and it employs a set of defined rules to identify harmful network activities and alerts. In high-speed bandwidth environments, different rule sets vary in detection performance [17]. We evaluate the performance using Snort Registered (SR), Snort Community (SC), and Emerging Threats (ET) rule sets.

5.2 Attack Call Flow Graph Analysis

To better understand the attack patterns and find the key point, we construct ACFG to assist in identifying and analyzing existing DDoS attacks, especially for the difference of distinctive paths between attack and benign traffic. We conduct kernel tracing on benign TCP traffic, which amounts to approximately 30GB in total. This benign flow serves as a reference baseline and assists us in gaining a deeper understanding of the impact of DDoS attacks on the kernel and in identifying characteristics of abnormal behavior.

Take TCP Connect Flood attack and the TCP SYN Flood attack as examples, as shown in Fig. 12 in Appendix. TCP Connect Flood attack potentially triggers several TCP connection management functions within the kernel, such as `tcp_rcv_established()` and `tcp_time_wait()`, among others. This class of attack results in a large influx of connection requests, causing the server to

Table 1. Drop nodes counts in various states

TCP-based DDoS Attacks	State	Count	TCP-based DDoS Attacks	State	Count
TCP Connect Flood	LISTEN	1	TCP SYN Flood	LISTEN	7
	SYN_RCVD	2		SYN_RCVD	12
	ESTABLISHED	5		ESTABLISHED	44
	ALL STATES	-		ALL STATES	3
TCP ACK Flood	LISTEN	6	TCP RST Flood	LISTEN	2
	SYN_RCVD	11		SYN_RCVD	1
	ESTABLISHED	18		ESTABLISHED	10
	ALL STATES	3		ALL STATES	1
ACK-PSH Flood	LISTEN	6	SYN-RST-ACK Flood	LISTEN	2
	SYN_RCVD	11		SYN_RCVD	4
	ESTABLISHED	21		ESTABLISHED	2
	ALL STATES	2		ALL STATES	-

continuously attempt to allocate resources to handle these requests, ultimately leading to resource exhaustion. TCP SYN Flood attacks trigger a large number of invocations of the `tcp_syn_ack_timeout()` function within the TCP stack. This function defines the timeout period during which the server waits for the client to respond with an ACK after sending a SYN-ACK response. In TCP SYN Flood attack scenario, the TCP state machine remains in the SYN_RCVD state and cannot progress to the next state.

We consider resolve oracle in the TCP LISTEN, SYN_RCVD, and ESTABLISHED states, given these states cover the complete window of the server side in the TCP three-way handshake. Such a way of focusing on core state transitions simplifies the state machine, making symbolic execution more efficient. Thus, in other TCP states, such as CLOSE_WAIT, the server will not accept any further packets. For different attacks, we mark different drop nodes in the source code. S2E symbolically executes Linux binary files, so these points will be mapped to the binary level. The number of drop nodes corresponding to different attacks is shown in Table 1. Due to space limitations, we have placed the original address table of drop nodes in Table 2 of the Appendix.

5.3 Symbol Execution Experiment Setup

Symbolic execution may get stuck at the beginning of execution and hard to reach deep paths, which is caused by path selection heuristic methods. Therefore, the key of the symbolic execution phase lies in the construction of TCP seed packets and the definition of pruning strategies. TCP seed packets guide the program along the path to the parts of the kernel and create side branches. Once construction process of the main path is completed, S2E explores side branches in depth. The promising seed packets help us penetrate deeper into the TCP stack quickly.

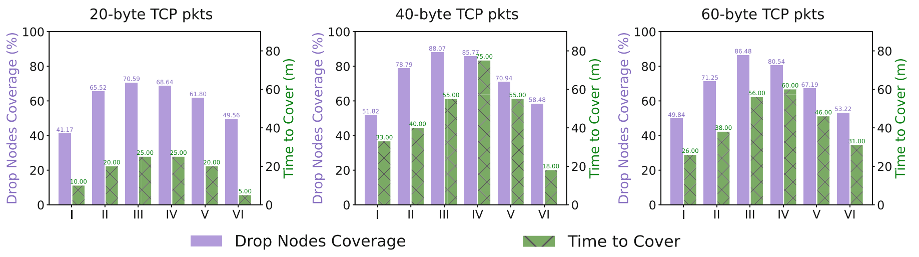

Fig. 7. Time costs and drop nodes coverage in symbolic execution.

In symbolic execution, we discard uninteresting paths, including: (i) Redundant paths that re-explore the same parts of the kernel code, *i.e.,* if a program path is identified to a previously explored path and reaches the same address with the same symbolic constraints, it will continue executing the same next branch. As a result, it can be discarded. (ii) Error detection path branches caused by incorrect concrete values generated by the solver. (iii) Path branches caused by Linux kernel check failures. (iv) Path branches leading to drop nodes. These strategies help us reduce the symbolic state space.

Although we generate specific TCP seed packets to reach drop nodes' addresses and employ optimization strategies to narrow down the search space for solutions, the complexity of the TCP stack makes it challenging for symbolic execution to provide comprehensive coverage. We examine the path coverage of the stack and track the accessed drop nodes, which can ensure critical nodes associated with potential attack paths have been checked.

In experiments, we send three types of symbolic data packets: 20-byte packets, 40-byte packets, and 60-byte packets, each of which includes a TCP header and payload. We observe that the composition of seed packet fields significantly affects the time cost of symbolic execution, especially when handling 40-byte and 60-byte packets, as shown in Fig. 7. We label six categories of attacks using Roman numerals: TCP Connect Flood (I), TCP SYN Flood (II), TCP ACK Flood (III), TCP RST Flood (IV), ACK-PSH Flood (V), and SYN-RST-ACK Flood (VI).

Large seed packets contain more variables and data to be symbolized, resulting in a significant increase in the number of paths that the symbolic execution engine needs to explore. This is because every possible branch of each conditional statement needs to be considered. It needs to backtrack to a previous branching point and reselect a path when the constraint is unsatisfied. These packets not only introduce more code blocks, increasing the time cost of symbolic execution but also imply exploring more paths to achieve higher code coverage.

After approximately 20 h of symbolic execution exploration, we utilize six instances to explore over 100,000 execution paths. During this process, more than 4,000 state transitions were triggered. Due to the server-side TCP state machine experiencing numerous repetitive cycles of state sequence (such as LISTEN $\rightarrow$ SYN_RCVD $\rightarrow$ SYN_RCVD), these transitions include repeated states, covering approximately 2,000 lines of code. Subsequently, we conduct further analysis of

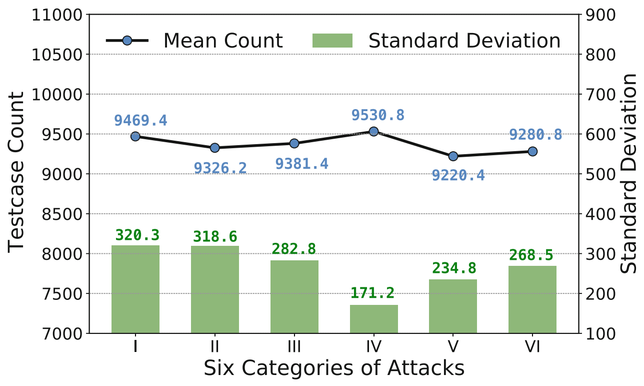

Fig. 8. Analysis of Testcase under various attacks.

the test cases associated with these newly discovered attack paths and perform kernel tracing. In subsequent experiments, we assess these new potential threats to better understand their potential risks and the impact of attacks.

5.4 Symbolic Execution Results

We solve the test cases for candidate attack sequences to generate specific values for TCP header fields and perform field padding and validation for the packets. Although selective symbolic execution explores paths close to the seed packets, the randomness of paths due to the complexity of the TCP protocol stack may lead to differences in path search results. Therefore, with the same configuration, we conduct five sets of experiments (labeled A to E) for six categories of attacks and record the following metrics:

Testcase Count. This metric represents the number of generated test cases. It demonstrates the exploration of potential attack paths by the symbolic execution engine and the quantity of generated attack samples.

Attack Success Rate. This metric reflects the ratio of actual successful attack attempts. It helps evaluate the effectiveness of the generated test cases in simulating real attacks.

CPU Utilization. By monitoring CPU utilization, we understand the system resource load during the attack.

Connection Queueing Rate. This metric shows the situation where TCP connection requests are forced to queue for processing due to the attack. This metric helps evaluate the performance of the system's responsiveness to service requests.

For the candidate attack sequence of each class attack, we conduct five independent simulations to observe the impact of different factors under the above metrics, as shown in Fig. 8 and 9. Actual data may varies due to network devices, configurations, and defense strategies in place traffic patterns can also influence the results. By monitoring and analyzing these metrics, we can more accurately assess the actual performance and effectiveness of the test cases generated by symbolic execution in the context of DDoS attack and defense.

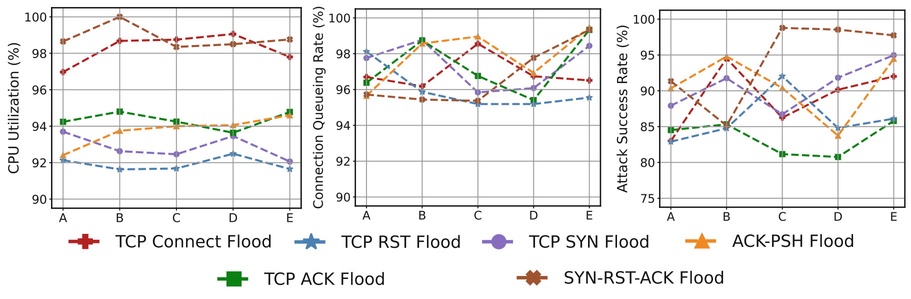

Fig. 9. Comparative metrics and impact of various attacks.

For the Testcase Count, the average number of Testcase Count is around 9300 for each type of attack, indicating that the symbolic execution engine explored attack paths to a similar extent across various types of attacks. The curves in Fig. 8 represent the average of five independent experiments for each type of attack, and the bars represent the standard deviation for each type of attack. From the averages, ACK-PSH Flood attack (V) and SYN-RST-ACK Flood attack (VI) had fewer test cases, indicating that the symbolic execution engine encountered fewer variants when exploring paths for these two types of attacks. The standard deviation indicates that the TCP RST Flood attack (IV) has the minimum fluctuation in test case count across different experiments.

Under different attacks, CPU utilization, connection queueing rate, and attack success rate show different performances as shown in Fig. 9. TCP Connect Flood and SYN-RST-ACK Flood attacks generally introduced more computation and connection requests, leading to higher CPU utilization, which was above 96% in experiments, even reaching 100%. Although TCP SYN Flood and TCP RST Flood attacks involve a large number of connection requests, their attack processes are simpler, thus requiring less CPU resources. TCP ACK Flood and ACK-PSH Flood attacks show moderate CPU utilization.

For the Connection Queueing Rate, the six attacks do not show significant differences. Effective test cases in most evaluations cause system capacity insufficiency and queuing of connection requests. This indicates that categories of attacks effectively exhaust server resources.

For the Attack Success Rate, TCP ACK Flood and TCP RST Flood attacks show lower success rates because these two types of attacks require matching valid connection states to deceive the TCP protocol stack, otherwise they would not affect the establishment of new connections or the continuation of existing ones. In contrast, attacks like TCP SYN Flood or TCP Connect Flood directly target the initialization process of connections, quickly filling the server's half-open connection queue and preventing the establishment of new legitimate connections. This directly affects server availability, hence they have higher attack success rates. ACK-PSH Flood and SYN-RST-ACK Flood attacks increase server processing load, leading to connection interruptions or service delays.

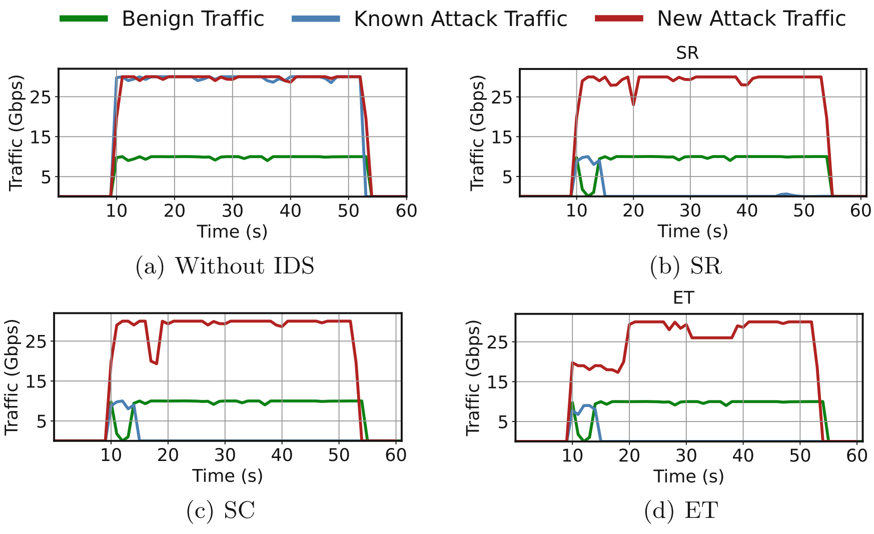

Fig. 10. Analysis of Benign vs. Known attack vs. New attack TCP traffic.

5.5 Evasion Evaluation Against IDS

We utilize three rule sets compatible with the Snort: SR, SC, and ET, and evaluate about 10,000 candidate attack sequences generated by symbolic execution.

We analyze the network traffic without IDS and then apply the SR, SC, and ET rule sets for attack detection. To better understand this detection and filtering process, we visualize this process, as shown in Fig. 10. The Fig. 10(a) shows the states of benign traffic, known attack traffic, and new attack traffic when no IDS checks are enabled. The other three figures in Fig. 10 are the traffic conditions using the SR, SC, and ET rule sets. It shows that the ET rule set has better detection performance than the other two.

Under baseline conditions without IDS inspection enabled, we first verify known attack traffic, which mainly consists of traffic in public datasets. The results show that when rule inspection is enabled, the defense system can effectively mark and filter known attack traffic. However, for new attack traffic generated by candidate attack sequences through symbolic execution, although Snort successfully detects and blocks some attacks, a significant amount of traffic bypasses Snort's detection and successfully reaches its intended victim.

We evaluate CPU Utilization and Connection Queueing Rate as mentioned above, confirming they indeed caused resource occupancy and TCP connection queue congestion. Figure 11 shows the state of system resources is saturated during new attacks and queuing of TCP connection requests caused by the limitation of CPU capacity.

In the first 15 s, CPU occupancy is low, maintaining around 20%, and the message queuing rate is nearly zero, indicating the system is operating normally without significant load or congestion. However, after 15 s, CPU occupancy

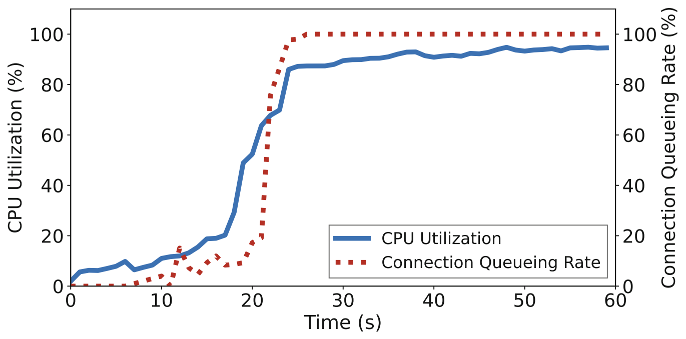

Fig. 11. Changes in CPU Utilization and Connection Queueing Rate over time.

began to rise sharply, quickly approaching 100%, indicating the attack started to cause significant processing pressure.

Meanwhile, the queuing rate of TCP connections began to rise slowly. After the CPU utilization reaches its peak, the message queuing rate gradually increases and reaches 100% at around 25 s. This means all new TCP connection requests are queuing up for processing, *i.e.,* unable to handle any TCP connection immediately. In this state, the system could not handle more load, and new requests could only wait, leading to service timeouts.

We found despite setting defense levels for the known six categories of attacks in the IDS, attackers can evade detection by adopting various attack variants, aiming to increase attack success or reduce detection risk. These variants include parameter randomization, attack mixing, malware use, IP address spoofing, attack segmentation, and evasion of known rules.

Further, we discover that successful evasion strategies are related to TCP timestamp option verification. Packets with time stamp echo reply (TSecr) not matching timestamp value (TSval) are usually detected and reported by rules, but successful IDS evasion cases with timestamped packet sequences include:

Insert Invalid Values. Randomizing Flag fields and TSval not conforming to the normal time progression pattern (e.g., echoing a value in the TSecr field that was never sent before in TSval), making packets appear as benign, delayed arrivals.

Imitate Normal Communication. Attackers observe normal TCP traffic timestamp patterns and imitate these patterns, using specific modes (e.g., adding extra microseconds every few packets) to alter the normal progression of timestamps.

Timestamp Obfuscation. Some sequence timestamps are abnormal, with their parsing observed to be incorrect on target servers. Compared to known attack traffic, IDS does not report these packets, but their TIME_WAIT state often last twice as long as the Maximum Segment Lifetime (MSL). Also, part of the timestamp confusion packets lead to abnormal transitions in the TCP state machine.

These attacks affect the TCP Retransmission Queue, with abnormal TSval causing the TCP stack to misjudge network conditions, affecting the calculation of retransmission timeouts. Multiple timers and timeout mechanisms in the TCP stack, like Keepalive timers and TIME_WAIT state processing, are also disrupted, affecting the TCP states.

Attackers could exploit flaws in TCP timestamp verification to disguise attack traffic as normal, avoiding detection by IDS. This camouflage method tampers with timestamps or inserts erroneous ones in attack packets, further increasing detection complexity and reducing accuracy. Although attacks using TCP timestamp options are not common, they are significantly effective. Developing IDS capable of deep analysis will help increase detection accuracy and reduce false positives.

6 Conclusion

In this work, we introduce DDoSMiner, an automated framework which utilizes the ACFG to extract critical attack points. In addition, we explore the system-level performance of an attack and provide visualization results. DDoSMiner integrates symbolic execution to systematically explore DDoS traffic variants with the guidance of identified key states in the TCP state machine. Our experiments generate a total of 9,741 candidate attack traffic variants, which are evaluated on the popular NIDS Snort with three main defense rule set toolkits. The result identifies one new attack traffic which is capable of bypassing all three defense measures, demonstrating the effectiveness of DDoSMiner in uncovering TCP-based DDoS attacks. Our work not only reveals potential security threats in the TCP protocol but also provides a new perspective on attack methodology, assisting researchers in better understanding and preventing network attacks.

7 Limitations and Future Work

Due to the complexity of the Linux kernel, exploration based on white-box strategies still has the problem of inefficiency. Distributed strategies can be considered to improve the efficiency of system operation. In addition, we chose a specific version of the Linux kernel v4.9.3 to evaluate our system. TCP state machines for other kernel versions and categories of attacks can be built through patch installation and other methods. Moreover, although we choose three widely covered and mature IDS rule sets for verification against new threats, specific subsequent defensive strategies and measures still need further research and exploration. In the future, we plan to improve DDoSMiner by increasing the path coverage of detection and applying DDoSMiner to other TCP stacks. Furthermore, DDoSMiner could be extended to explore more protocols. For different protocols, different drop nodes can be designed for expansion.

Acknowledgements. This work was supported in part by National Natural Science Foundation of China (62227805, 62072398 and 62172405), by SUTD-ZJU IDEA

Grant for visiting professors (SUTD-ZJUVP201901), by the Natural Science Foundation of Jiangsu Province (BK20220075), by the Fok Ying-Tung Education Foundation for Young Teachers in the Higher Education Institutions of China (20193218210004), by State Key Laboratory of Mathematical Engineering and Advanced Computing, and by Key Laboratory of Cyberspace Situation Awareness of Henan Province (HNTS2022001).

A Visualization and Analysis of System Calls

The ACFG extracted from the TCP Connect Flood and TCP SYN Flood attacks are shown in Fig. 12. The nodes and edges of ACFG are highlighted in different colors to represent the corresponding types of packets (benign or attack). In the figure, green-colored elements represent syscalls triggered by benign packets, orange-colored elements represent syscalls triggered by attack packets, and blue elements represent syscalls triggered by both types of packets. The red colored nodes are identified as Pivotal Nodes. According to the definition in Sect. 3.2, the following nodes represent the change of TCP state:

(i) `tcp_v4_syn_recv_sock`: This is a critical function for handling client SYN packets. This function checks the current TCP state (for example, whether it is in LISTEN state) to determine if the connection can be established.
(ii) `tcp_check_req`: This function checks whether the SYN packet is valid and whether there are resources available to handle this new connection request. If the SYN packet is invalid, a RST packet will be sent to refuse the connection.
(iii) `tcp_v4_do_rcv`: When the client sends an ACK packet in response to the server's SYN and ACK, this function processes the ACK packet, thereby advancing the connection state transition process.
(iv) `tcp_rcv_state_process`: This function is crucial in the TCP state machine. Within this function, if the current connection state is SYN_RCVD and an appropriate ACK segment is received, the state transitions to ESTABLISHED. Other state transitions in the TCP connection and the processing of related packets also call this function.
(v) `tcp_rcv_established`: This function handles inputs in the ESTABLISHED state.
(vi) `tcp_close`: This function is used to close a TCP connection. It releases the resources occupied by the connection and changes the connection state.

By comparing the orange nodes in the syscall of TCP Connect Flood and TCP SYN Flood attacks, we can see that the two attacks have different characteristics (detailed analysis in Sect. 5.2).

B The Kernel Address Corresponding to the Full Drop Nodes for Six Categories of Attacks

We extract the potential drop nodes from the ACFG and then indexed the corresponding addresses in the kernel. These addresses serve as path termination

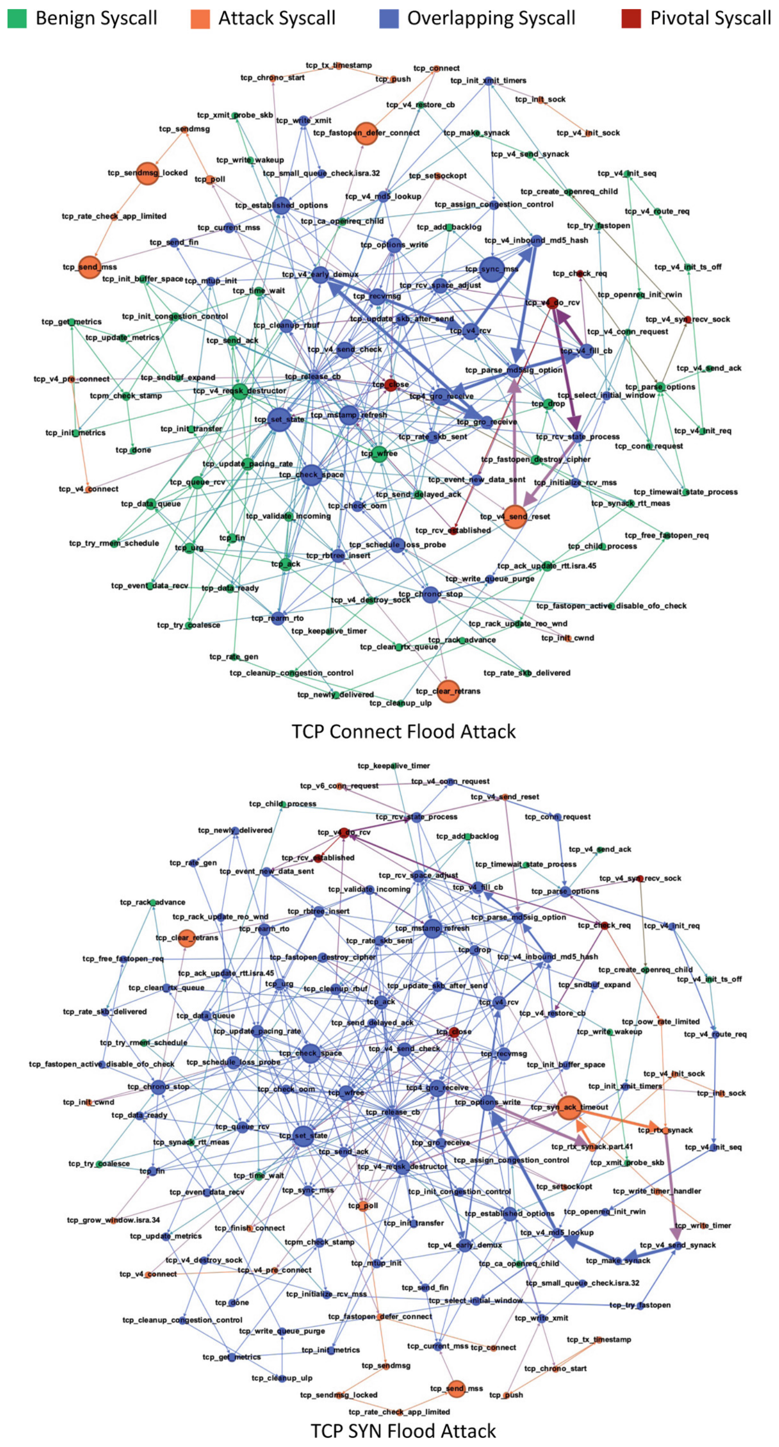

Fig. 12. Visualization of system calls for TCP Connect Flood Attack and TCP SYN Flood Attack.

Table 2. Kernel addresses associated with drop nodes for different attacks

TCP-based DDoS Attacks	Address			
TCP Connect Flood	ffffffff819dc550	ffffffff819d93d0	ffffffff819e04d0	ffffffff819e9bf0
	ffffffff819dc3a0	ffffffff819f1910	ffffffff819ef4d0	
TCP SYN Flood	ffffffff819f63b0	ffffffff819e5410	ffffffff819eaee0	ffffffff819eda80
	ffffffff819d9ce0	ffffffff819de910	ffffffff819ead60	ffffffff819de6e0
	ffffffff819d9e30	ffffffff819e63d0	ffffffff819de950	ffffffff819d2250
	ffffffff819d5010	ffffffff819e4dd0	ffffffff819f5720	ffffffff819eb9d0
	ffffffff819f6330	ffffffff819d1830	ffffffff819f5790	ffffffff819d3290
	ffffffff819dc3a0	ffffffff819d9ef0	ffffffff819e91a0	ffffffff819e2a70
	ffffffff819da3e0	ffffffff819f5a90	ffffffff819eafe0	ffffffff819ebbf0
	ffffffff819de6a0	ffffffff819d38d0	ffffffff819e4600	ffffffff819ea720
	ffffffff819d9e60	ffffffff819d19d0	ffffffff819e4550	ffffffff819dee80
	ffffffff819d9f50	ffffffff819eca70	ffffffff819d1c00	ffffffff819ea5d0
	ffffffff819dbee0	ffffffff819ed670	ffffffff819d7890	ffffffff819d1790
	ffffffff819d3490	ffffffff81a87d40	ffffffff819d6af0	ffffffff819f5440
	ffffffff819f25c0			
TCP ACK Flood	ffffffff819e5410	ffffffff819de6a0	ffffffff819d19d0	ffffffff819eca70
	ffffffff819de910	ffffffff819d1830	ffffffff819f5790	ffffffff819ef4d0
	ffffffff819e63d0	ffffffff819d38d0	ffffffff819d1c00	ffffffff819ed670
	ffffffff819f5440	ffffffff819d3490	ffffffff819eda80	ffffffff819d3290
	ffffffff819d7890	ffffffff819d6af0		
TCP RST Flood	ffffffff819d9470	ffffffff819d9080	ffffffff819efb70	ffffffff819f06c0
	ffffffff819f6a30	ffffffff819e2b60	ffffffff819f0560	ffffffff819f6d10
	ffffffff819ebfe0	ffffffff819ee750		
ACK-PSH Flood	ffffffff819e5410	ffffffff819e2a70	ffffffff819d38d0	ffffffff819eb9d0
	ffffffff819de910	ffffffff819dc3a0	ffffffff819d19d0	ffffffff819ef4d0
	ffffffff819e63d0	ffffffff819de6a0	ffffffff819f5790	ffffffff819eda80
	ffffffff819f5440	ffffffff819d1830	ffffffff819e4600	ffffffff819ebbf0
	ffffffff819d3290	ffffffff819e0460	ffffffff819eca70	ffffffff819d6af0
	ffffffff819ea5d0	ffffffff819d3490	ffffffff819ed670	ffffffff819d1c00
	ffffffff819e4550	ffffffff819d7890	ffffffff81a87d40	
SYN-RST-ACK Flood	ffffffff819e0460	ffffffff819ea5d0	ffffffff819ef4d0	ffffffff81a87d40
	ffffffff819e4600	ffffffff819e4550		

points for symbolic execution. The detailed attack types and the corresponding address we extracted for the experiment are listed in Table 2.

References

1. Agrawal, N., Tapaswi, S.: Defense mechanisms against ddos attacks in a cloud computing environment: state-of-the-art and research challenges. IEEE Commun. Surv. Tutorials **21**(4), 3769–3795 (2019)
2. Antonakakis, M., April, T., et al.: Understanding the mirai botnet. In: 26th USENIX Security Symposium (USENIX Security 17), pp. 1093–1110 (2017)
3. Baig, Z.A., et al.: Controlled access to cloud resources for mitigating economic denial of sustainability (edos) attacks. Comput. Netw. **97**, 31–47 (2016)
4. Baldoni, R., Coppa, E., et al.: A survey of symbolic execution techniques. ACM Comput. Surv. (CSUR) **51**(3), 1–39 (2018)
5. Bhale, P., Chowdhury, D.R., Biswas, S., Nandi, S.: Optimist: Lightweight and transparent ids with optimum placement strategy to mitigate mixed-rate ddos attacks in iot networks. IEEE Internet of Things Journal (2023)
6. Bock, K., et al.: Weaponizing middleboxes for {TCP} reflected amplification. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 3345–3361 (2021)
7. Cadar, C., Dunbar, D., Klee, D.E.: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proceedings of Operating System Design and Implementation, pp. 209–224
8. Cadar, C., Sen, K.: Symbolic execution for software testing: three decades later. Commun. ACM **56**(2), 82–90 (2013)
9. Chang, R.K.: Defending against flooding-based distributed denial-of-service attacks: a tutorial. IEEE Commun. Mag. **40**(10), 42–51 (2002)
10. Chipounov, V., Kuznetsov, V., Candea, G.: S2e: a platform for in-vivo multi-path analysis of software systems. Acm Sigplan Notices **46**(3), 265–278 (2011)
11. Chipounov, V., et al.: The s2e platform: design, implementation, and applications. ACM Trans. Comput. Syst. (TOCS) **30**(1), 1–49 (2012)
12. Deshmukh, R.V., Devadkar, K.K.: Understanding ddos attack & its effect in cloud environment. Proc. Comput. Sci. **49**, 202–210 (2015)
13. Doshi, R., Apthorpe, N., Feamster, N.: Machine learning ddos detection for consumer internet of things devices. In: 2018 IEEE Security and Privacy Workshops (SPW), pp. 29–35. IEEE (2018)
14. Fayaz, S.K., Tobioka, Y., et al.: Bohatei: Flexible and elastic {DDoS} defense. In: 24th USENIX Security Symposium (USENIX Security 15), pp. 817–832 (2015)
15. Feamster, N., et al.: The road to sdn: an intellectual history of programmable networks. ACM SIGCOMM Comput. Commun. Rev. **44**(2), 87–98 (2014)
16. Gaurav, A., Gupta, B.B., Alhalabi, W., Visvizi, A., Asiri, Y.: A comprehensive survey on ddos attacks on various intelligent systems and it's defense techniques. Int. J. Intell. Syst. **37**(12), 11407–11431 (2022)
17. Granberg, N.: Evaluating the effectiveness of free rule sets for snort (2022)
18. Guha, B., Mukherjee, B.: Network security via reverse engineering of tcp code: vulnerability analysis and proposed solutions. IEEE Netw. **11**(4), 40–48 (1997)
19. Herrera, J.G., Botero, J.F.: Resource allocation in nfv: a comprehensive survey. IEEE Trans. Netw. Serv. Manage. **13**(3), 518–532 (2016)
20. Hong, S., Xu, L., et al.: Poisoning network visibility in software-defined networks: New attacks and countermeasures. In: Network and Distributed System Security Symposium (2015). https://api.semanticscholar.org/CorpusID:12312831
21. Jin, C., Wang, H., Shin, K.G.: Hop-count filtering: an effective defense against spoofed ddos traffic. In: Proceedings of the 10th ACM Conference on Computer and Communications Security, pp. 30–41 (2003)

22. Joseph, D.A., et al.: A policy-aware switching layer for data centers. In: Proceedings of the ACM SIGCOMM 2008 Conference On Data Communication, pp. 51–62 (2008)
23. Kaur, R., Singh, M.: A survey on zero-day polymorphic worm detection techniques. IEEE Commun. Surv. Tutorials **16**(3), 1520–1549 (2014)
24. Keromytis, A.D., et al.: Sos: an architecture for mitigating ddos attacks. IEEE J. Sel. Areas Commun. **22**(1), 176–188 (2004)
25. Krupp, J., Grishchenko, I., Rossow, C.: {AmpFuzz}: Fuzzing for amplification {DDoS} vulnerabilities. In: 31st USENIX Security Symposium (USENIX Security 22), pp. 1043–1060 (2022)
26. Lalou, M., Tahraoui, M.A., Kheddouci, H.: The critical node detection problem in networks: a survey. Comput. Sci. Rev. **28**, 92–117 (2018)
27. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines-a survey. Proc. IEEE **84**(8), 1090–1123 (1996)
28. Liu, Z., et al.: Jaqen: A {High-Performance}{Switch-Native} approach for detecting and mitigating volumetric {DDoS} attacks with programmable switches. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 3829–3846 (2021)
29. Liu, Z., Jin, H., Hu, Y.C., Bailey, M.: Practical proactive ddos-attack mitigation via endpoint-driven in-network traffic control. IEEE/ACM Trans. Network. **26**(4), 1948–1961 (2018)
30. Mirsky, Y., Guri, M.: Ddos attacks on 9-1-1 emergency services. IEEE Trans. Dependable Secure Comput. **18**(6), 2767–2786 (2020)
31. Mizrak, A.T., Savage, S., Marzullo, K.: Detecting compromised routers via packet forwarding behavior. IEEE Netw. **22**(2), 34–39 (2008)
32. Moore, D., Shannon, C., Brown, D.J., Voelker, G.M., Savage, S.: Inferring internet denial-of-service activity. ACM Trans. Comput. Syst. (TOCS) **24**(2), 115–139 (2006)
33. Mosenia, A., Jha, N.K.: A comprehensive study of security of internet-of-things. IEEE Trans. Emerg. Top. Comput. **5**(4), 586–602 (2016)
34. Nayak, J., Meher, S.K., Souri, A., Naik, B., Vimal, S.: Extreme learning machine and bayesian optimization-driven intelligent framework for iomt cyber-attack detection. J. Supercomput. **78**(13), 14866–14891 (2022)
35. Nazario, J.: Ddos attack evolution. Netw. Secur. **2008**(7), 7–10 (2008)
36. O'Leary, M., O'Leary, M.: Snort. Cyber Operations: Building, Defending, and Attacking Modern Computer Networks, pp. 605–641 (2015)
37. Praseed, A., Thilagam, P.S.: Multiplexed asymmetric attacks: Next-generation ddos on http/2 servers. IEEE Trans. Inf. Forensics Secur. **15**, 1790–1800 (2019)
38. Rossow, C.: Amplification hell: Revisiting network protocols for ddos abuse. In: 2014 Network and Distributed System Security Symposium (2014)
39. Santanna, J.J., van Rijswijk-Deij, R., et al.: Booters-an analysis of ddos-as-a-service attacks. In: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 243–251. IEEE (2015)
40. Sharafaldin, I., Lashkari, A.H., Hakak, S., Ghorbani, A.A.: Developing realistic distributed denial of service (ddos) attack dataset and taxonomy. In: 2019 International Carnahan Conference on Security Technology (ICCST), pp. 1–8. IEEE (2019)
41. Shiravi, A., Shiravi, H., Tavallaee, M., Ghorbani, A.A.: Toward developing a systematic approach to generate benchmark datasets for intrusion detection. Comput. Secur. **31**(3), 357–374 (2012)

42. Shoshitaishvili, Y., Wang, R., et al.: Sok:(state of) the art of war: offensive techniques in binary analysis. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 138–157. IEEE (2016)
43. Song, H., Liu, J., Yang, J., Lei, X., Xue, G.: Two types of novel dos attacks against cdns based on http/2 flow control mechanism. In: European Symposium on Research in Computer Security, pp. 467–487. Springer (2022)
44. Song, Z., Zhao, Z., Zhang, F., et al.: I2RNN: An incremental and interpretable recurrent neural network for encrypted traffic classification. IEEE Transactions on Dependable and Secure Computing (2023)
45. Specht, S., Lee, R.: Taxonomies of distributed denial of service networks, attacks, tools and countermeasures. CEL2003-03, Princeton University, Princeton, NJ, USA (2003)
46. Srivastava, A., Gupta, B.B., Tyagi, A., Sharma, A., Mishra, A.: A recent survey on ddos attacks and defense mechanisms. In: Nagamalai, D., Renault, E., Dhanuskodi, M. (eds.) Advances in Parallel Distributed Computing: First International Conference on Parallel, Distributed Computing Technologies and Applications, PDCTA 2011, Tirunelveli, India, September 23-25, 2011. Proceedings, pp. 570–580. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24037-9_57
47. Stephens, N., Grosen, J., et al.: Driller: Augmenting fuzzing through selective symbolic execution. In: NDSS. vol. 16, pp. 1–16 (2016)
48. Sung, M., Xu, J.: Ip traceback-based intelligent packet filtering: a novel technique for defending against internet ddos attacks. IEEE Trans. Parallel Distrib. Syst. **14**(9), 861–872 (2003)
49. Thing, V.L., Sloman, M., Dulay, N.: Non-intrusive ip traceback for ddos attacks. In: Proceedings of the 2nd ACM Symposium On Information, Computer and Communications Security, pp. 371–373 (2007)
50. Wagner, D., Kopp, D., et al.: United we stand: Collaborative detection and mitigation of amplification ddos attacks at scale. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 970–987 (2021)
51. Wang, Z., Zhu, S.: Symtcp: Eluding stateful deep packet inspection with automated discrepancy discovery. In: Network and Distributed System Security Symposium (NDSS) (2020)
52. Xing, J., Wu, W., Chen, A.: Ripple: A programmable, decentralized {Link-Flooding} defense against adaptive adversaries. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 3865–3881 (2021)
53. Yaar, A., Perrig, A., Song, D.: Stackpi: new packet marking and filtering mechanisms for ddos and ip spoofing defense. IEEE J. Sel. Areas Commun. **24**(10), 1853–1863 (2006)
54. Yan, Q., et al.: Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges. IEEE Commun. Surv. Tutorials **18**(1), 602–622 (2015)
55. Yoachimik, O., Pacheco, J.: DDoS threat report for 2023 q2 (2023). https://blog.cloudflare.com/ddos-threat-report-2023-q2/ Accessed 20 Sept 2023
56. Zargar, S.T., Joshi, J., Tipper, D.: A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. IEEE Commun. Surv. Tutorials **15**(4), 2046–2069 (2013)
57. Zhang, M., Li, G., et al.: Poseidon: mitigating volumetric ddos attacks with programmable switches. In: the 27th Network and Distributed System Security Symposium (NDSS 2020) (2020)

58. Zhang, Z., Yuan, B., Yang, K., Zou, D., Jin, H.: Statediver: Testing deep packet inspection systems with state-discrepancy guidance. In: Proceedings of the 38th Annual Computer Security Applications Conference, pp. 756–768 (2022)
59. Zhao, Z., Li, Z., et al.: DDoS Family: A Novel Perspective for Massive Types of DDoS Attacks. Comput, Secur (2023)
60. Zhao, Z., Li, Z., et al.: ERNN: error-resilient RNN for encrypted traffic detection towards network-induced phenomena. IEEE Transactions on Dependable and Secure Computing (2023)
61. Zhao, Z., Liu, Z., et al.: Effective DDoS mitigation via ML-driven in-network traffic shaping. IEEE Transactions on Dependable and Secure Computing (2024)
62. Zhao, Z., et al.: CMD: co-analyzed iot malware detection and forensics via network and hardware domains. IEEE Transactions on Mobile Computing (2023)
63. Zou, Y.H., Bai, J.J., et al.: {TCP-Fuzz}: Detecting memory and semantic bugs in {TCP} stacks with fuzzing. In: 2021 USENIX Annual Technical Conference (USENIX ATC 21), pp. 489–502 (2021)