ACCEPTED FROM OPEN CALL

PacketScope: Bridging eBPF and LLMs for In-Stack Defense Against Subtle TCP/IP Exploits

Xuewei Feng, Qixun Tang, Min Li, Yuxiang Yang, Zhaoxi Li, Xingxiang Zhan, Yi He, Chi Chen, Qi Li, and Ke Xu

Abstract

The TCP/IP protocol suite underpins today's Internet infrastructure, yet emerging attacks increasingly exploit subtle semantic flaws in protocol interactions to bypass traditional defenses and inflict severe damage. In this article, we introduce `PacketScope`, a framework that integrates kernel-level observability via the extended Berkeley Packet Filter (eBPF) with reasoning and policy synthesis driven by large language models (LLMs), enabling comprehensive and effective in-stack defense against a wide range of subtle TCP/IP exploits. Specifically, `PacketScope` follows a three-stage workflow. First, leveraging eBPF, it performs per-packet inspection within the kernel to construct protocol interaction graphs (PIGs) across layers and protocols, capturing long-context semantics and fine-grained session dynamics. Second, it employs an LLM to reason over PIGs, identifying anomalous interactions and generating enforcement policies expressed as eBPF rules. Finally, these rules are enforced in-kernel for real-time blocking, mitigating attacks directly within the TCP/IP stack. We implement a Linux prototype that instruments critical TCP/IP components with eBPF hooks and integrates LLMs (e.g., DeepSeek, TrafficLLM) for semantic analysis. Evaluation with real-world traffic traces, synthetic attack scenarios, and stress-test environments involving mixed protocol interactions shows that `PacketScope` detects a wide spectrum of attacks — including cross-protocol interference, state desynchronization, and covert connection manipulation — while incurring less than 2.5% performance loss. Unlike existing packet- or flow-level approaches based on signatures or classification, `PacketScope` demonstrates that coupling kernel-level observability with AI-driven semantic reasoning enables practical, effective, and lightweight in-stack defenses.

Introduction

The TCP/IP protocol suite serves as the backbone of the Internet infrastructure, providing the essential foundation for seamless data exchange across heterogeneous networks worldwide. Yet, its ubiquity and critical role also make it an attractive and persistent target for cyberattacks [1]. In particular, subtle vulnerabilities stemming from misalignments or inconsistencies during protocol interactions present serious security risks. As illustrated in Fig. 1, end-to-end communication between a server and a client involves a series of complex interactions: DNS (Domain Name System) lookups to resolve domain names, cross-layer processing within the server's TCP/IP stack, inter-domain routing to deliver packets across the Internet, and even middlebox translations that reshape packet semantics. An off-path attacker can deliberately trigger and exploit vulnerabilities during these complex protocol interactions to compromise communication integrity, leading to severe disruptions and undermining the trustworthiness of Internet connectivity.

For instance, an off-path attacker may induce desynchronization between the TCP and IP layers of a victim server, causing unintended IP fragmentation of TCP segments. This allows the attacker to bypass the Path MTU Discovery (PMTUD) mechanism (RFC 1191, RFC 1812) and mount IP fragmentation-based exploits against TCP [1]. Similarly, attackers can exploit weaknesses in the IP Identification (IPID) field to hijack upper-layer TCP connections, a threat observed in more than 20% of popular web servers on the Internet [1], or exploit weaknesses in ICMP error handling to compromise upper-layer DNS sessions, affecting 85% of popular DNS services on the Internet [2, 3]. Moreover, semantic gaps and information forgery across protocol interactions enable large-scale off-path traffic manipulation. Real-world evaluations on the Internet show that these vulnerabilities can be exploited to affect} over 90,000 public servers across 5,184 ASes, leading to denial-of-service (DoS) [1, 4], and to compromise more than 89% Wi-Fi networks with man-in-the-middle (MitM) exploits [1].

While considerable efforts have been made to investigate and mitigate TCP/IP exploits, it remains highly challenging to develop a general defense framework within the fundamental kernel layer that can identify and block such exploits in real time. The root difficulty lies in the stealthy

Xuewei Feng, Yuxiang Yang, and Zhaoxi Li are with Tsinghua University, China; Qixun Tang is with the Institute of Computing Technology, the Chinese Academy of Sciences and Zhongguancun Lab, China; Min Li and Xingxiang Zhan are with Zhongguancun Lab, China; Yi He is with Wuhan University, China; Chi Chen is with Meituan Corporation, China; Qi Li and Ke Xu (corresponding author) are with Tsinghua University and Zhongguancun Lab, China.

Digital Object Identifier: 10.1109/MCOM.001.2500660

nature of the exploits and the inherent difficulty of capturing them during protocol interactions at the kernel level. As a result, prior work has primarily resorted to ad hoc, patch-by-patch fixes [1, 5–7], which are costly to maintain, often require reboots or system reconfigurations to take effect, and may be impractical in scenarios where servers must provide persistent online services. These limitations highlight the urgent need for a systematic and adaptive framework to uncover and mitigate such subtle vulnerabilities.

In this article, we present `PacketScope`, a framework that integrates fine-grained kernel observability with LLM-driven reasoning to provide in-stack defenses against stealthy TCP/IP exploits. Leveraging eBPF (extended Berkeley Packet Filter, a Linux kernel mechanism for safely executing sandboxed programs in kernel space)), `PacketScope` captures precise execution traces of packet processing, including function-level invocation chains, timing information, and cross-layer dependencies. These detailed observations expose subtle semantic inconsistencies and anomalous protocol interactions that traditional logging or coarse-grained monitoring often fail to reveal. Complementing this, the LLM-driven reasoning component utilizes contextual knowledge of protocol semantics and prior exploit patterns to interpret the captured traces. By correlating raw kernel events with higher-level attack strategies, the reasoning engine can differentiate benign irregularities from adversarial manipulations.

Specifically, `PacketScope` includes the following three core components:

1. *Protocol Interaction Graph Construction*: We first extract a protocol interaction graph of each flow inside the kernel using eBPF. This graph captures the sequence of behaviors associated with the flow, including potential out-of-band packets crafted by an attacker and injected into the flow. Since these injected packets must share the same five-tuple to be accepted by the target flow, their anomalies can be explicitly revealed within the interaction graph.
2. *LLM-Guided Analysis and eBPF Rule Generation*: We then leverage these interaction graphs to prompt an LLM (e.g., TrafficLLM [8] or DeepSeek) to investigate abnormalities and infer potential vulnerabilities. Based on this reasoning, the LLM generates eBPF filtering or enforcement rules, which are compiled into eBPF maps and made ready for in-kernel enforcement.
3. *In-Kernel Enforcement and Real-Time Blocking*: Finally, the generated eBPF programs are dynamically loaded into the kernel, enabling in-stack filtering of malicious packets and preventing packet manipulation. In summary, `PacketScope` tightly integrates observability, reasoning, and enforcement into a unified framework, effectively bridging the gap between detection and defense in modern TCP/IP stack security.

We implement our defense on Linux kernel 6.8, demonstrating its ability to systematically mitigate semantic vulnerabilities in the TCP/IP suite. In our evaluation, `PacketScope` outperforms baseline systems, detecting a wide range of stealthy TCP/IP exploits — including ICMP-based path MTU manipulation, off-path hijacking via spoofed ICMP redirects, forged `RST`-based connection termination, DNS cache poisoning exploiting predictable UDP port allocation, TCP session hijacking via IPID side channels, and NAT-based DoS through malicious session mapping removal — while incurring less than 2.5% end-to-end throughput overhead. We release the source code of `PacketScope` at https://github.com/Internet-Architecture-and-Security/PacketScope, which includes the full framework implementation, environment setup instructions, and usage tutorials. To avoid ethical concerns, the real-world attack traces used in our evaluation are not publicly released, but can be made available upon reasonable request.

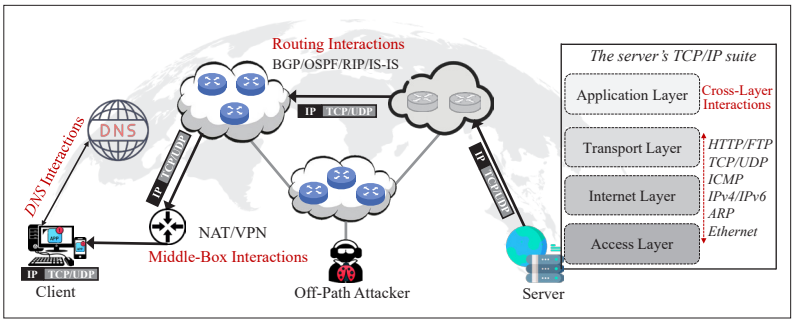

FIGURE 1. Cross-protocol interactions in end-to-end communications and the off-path attacker.

Background and Motivation

Modern TCP/IP exploits differ fundamentally from earlier attacks that target a single vulnerability or a specific protocol. Instead, they orchestrate sequences of stealthy, carefully timed manipulations spanning multiple layers and protocols. These manipulations exploit semantic gaps and misalignments arising from protocol interactions [1, 2, 3, 9]. Cross-layer and cross-protocol strategies — such as routing manipulations, ICMP/PMTUD abuse, UDP server routing manipulations, and DNS/transport interplay — enable off-path attacks that bypass defenses designed for isolated vulnerabilities.

In this article, we consider an off-path network attacker who can inject spoofed packets but does not control the communicating endpoints or the `PacketScope` system itself. Despite this limited capability, such attackers can still inflict severe damage on TCP/IP communications. For example, off-path TCP hijacking via path MTU discovery inconsistencies allows attackers to craft ICMP messages, exploit fragmentation behavior, and synchronize with TCP sequence number validation to stealthily inject malicious payloads [1]. Similarly, semantic gaps between UDP and ICMP can be leveraged to manipulate routing state and poison DNS caches, either by chaining ICMP error generation with UDP socket handling to drop legitimate traffic [1] or by predicting ephemeral UDP source ports to inject fake DNS replies [2, 3]. These exploits emerge from multi-phase interactions that appear legitimate in isolation but collectively grant adversarial control — a risk amplified by TCP/IP's layered architecture and subtle cross-protocol inconsistencies.

Such multi-step, cross-protocol exploits render traditional defenses largely ineffective. Signature-based intrusion detection systems [10, 11] fail to capture concealed logic dispersed across multiple protocol exchanges, while patch-by-patch kernel updates [5, 7] are reactive, costly, and disruptive to operations. Despite decades of incremental defenses, Internet infrastructure remains persistently

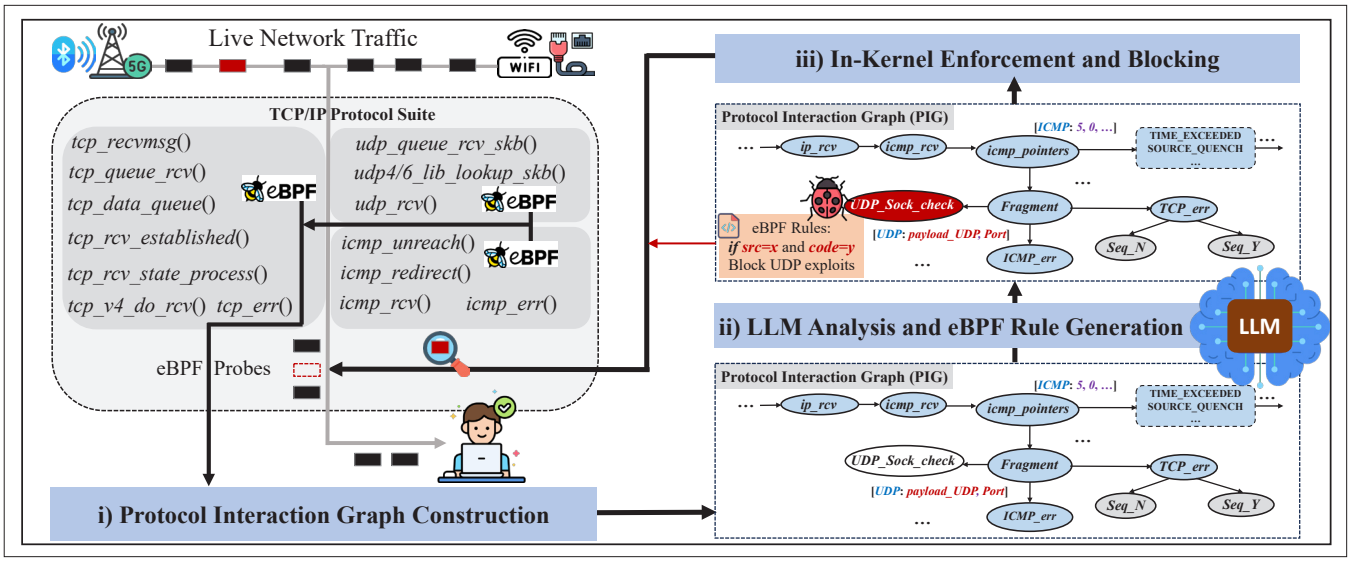

FIGURE 2. Design of `PacketScope` and its main components.

exposed to these concealed semantic vulnerabilities. These limitations motivate the development of a defense that unifies deep, real-time observability of packet processing with semantic reasoning across layers, enabling precise and low-latency mitigation.

Design and Methodology

The design of `PacketScope` is driven by the need to bridge high-level semantic reasoning with low-level kernel enforcement, thereby enabling real-time defense against sophisticated TCP/IP exploits. As illustrated in Fig. 2, `PacketScope` effectively identifies and blocks these subtle TCP/IP exploits through a three-stage pipeline:

- Extracting fine-grained protocol interaction graphs from live network traffic,
- Leveraging LLMs to interpret and reason about suspicious interaction patterns, and automatically generating eBPF enforcement rules,
- Enforcing these rules inside the kernel to enable low-latency blocking of ongoing and subsequent attacks.

This design balances semantic richness with efficiency: the LLM reasons about attacks in user space, while eBPF enforces countermeasures precisely in the kernel.

Protocol Interaction Graph Construction

Network traffic traces reveal only observable packet sequences, missing the underlying processing logic and intricate inter-dependencies across protocol layers. Therefore, the first step of `PacketScope` is the construction of a Protocol Interaction Graph (PIG), which provides a fine-grained, function-level view of how live packets are processed as they enter the TCP/IP stack. Each node in PIG represents a kernel function invoked during packet processing, while directed edges capture the causal relationships between successive calls, exposing execution semantics that are invisible from the wire-level perspective. By systematically modeling interactions across layers and modules, the PIG makes the implicit attack surface visible and analyzable, and serves as a structured foundation for subsequent LLM-based reasoning about potential attack behaviors, enabling precise understanding of complex, multi-layer exploits. To construct such PIGs, `PacketScope` leverages eBPF Kprobes (a debugging mechanism for the Linux kernel) and tracepoints (predefined hook points in the Linux kernel) to insert lightweight observation programs into critical functions across the IP, TCP, UDP, and ICMP subsystems. In Linux, these probes are programmatically registered via the BPF system call or through libraries like BCC (BPF Compiler Collection), specifying the target kernel function name (e.g., `ip_rcv()`, `tcp_v4_rcv()`, `tcp_v6_rcv()`, and `icmp_unreach()`) for kprobes, or the return point for kretprobes.

Once registered, the eBPF program executes in kernel space whenever the hooked function is invoked, allowing `PacketScope` to safely capture execution context without modifying the kernel. Each probe collects rich contextual information, encompassing function arguments, kernel data structures, and packet metadata such as 5-tuple, TCP sequence and acknowledgment numbers, payload length, segmentation flags, and NAT translation details. These data are stored in eBPF maps — high-performance key-value stores shared between kernel and user space — ensuring minimal overhead while enabling safe communication with user-space components. As shown in Algorithm 1, `PacketScope` maintains a per-flow call stack — identified by the packet 5-tuple and encompassing the corresponding packet sequence P — within the BPF program. This stack is crucial for reconstructing the causal and temporal relationships among function invocations. Upon function entry, a node representing the invocation is pushed onto the stack; upon return, the node is popped, and a directed edge is recorded from the caller to the callee. This approach ensures that even deeply nested or recursive kernel execution paths are faithfully captured in the PIG.

Each PIG node is labeled with the corresponding function name f and augmented with comprehensive metadata beyond standard packet headers, such as precise kernel timestamps, segmentation events, and NAT translation states, enabling fine-grained analysis of protocol interactions. Traces from multiple packets are merged incrementally to update the dynamic graph $G = (V, E)$, with directed edges reflecting the true func-

tion invocation order. This granular, context-aware representation enables the detection of subtle TCP/IP exploits spanning multiple layers or involving complex kernel logic, such as PMTUD manipulation, NAT semantic gaps, or TCP side channels.

For a concrete PMTUD and NAT scenario, consider a TCP packet traversing a NAT-enabled network where a Path MTU Discovery event occurs. The packet first triggers `ip_rcv` or `ipv6_rcv`, generating a PIG node representing packet reception and pushing it onto the per-packet stack. If NAT translation is required, the `nf_nat_ipv4_in` probe logs a second node, creating an edge from `ip_rcv` to `nf_nat_ipv4_in`. The packet then proceeds to the TCP layer, triggering probes such as `tcp_v4_rcv` or `tcp_ack`, generating additional nodes and edges according to function returns and the call stack. Subsequently, when a PMTUD-related ICMP message associated with this TCP flow is received, the `icmp_unreach` probe captures this event, logging ICMP type, code, original packet headers, and timestamp. A cross-layer edge links this node to the TCP function that initiated the original probe, completing the representation of the packet's journey. As illustrated in Fig. 3, the resulting PIG captures the full spectrum of protocol interactions — including NAT translation, TCP segmentation, ICMP feedback, and kernel function calls — enabling analysts to trace complex sequences, such as a TCP segment being dropped due to an incorrect sequence number after NAT translation. This fine-grained, contextual view of network behavior is critical for detecting subtle vulnerabilities and attacks in live systems.

LLM-Guided Analysis and eBPF Rule Generation

Once a protocol interaction graph (PIG) is constructed, the second stage leverages LLMs to perform deep semantic reasoning over protocol interactions, aiming to uncover concealed TCP/IP exploits. By treating the PIG as long-context input, the LLM can link early events in a flow to their delayed consequences, capturing dependencies that conventional short-window heuristics often miss. Beyond correlation, LLMs perform multi-step reasoning: they follow protocol branches, infer prerequisite conditions, cross-validate across layers, and expose semantic inconsistencies. This allows them to construct causal paths from local anomalies to global vulnerabilities. Even in the absence of explicit signatures, LLMs can abstract anomalous interaction patterns — such as asymmetric dependencies or information leakage chains — thereby generalizing beyond known exploits and exposing previously unseen attack surfaces. `PacketScope` supports general-purpose pre-trained LLMs (e.g., DeepSeek-67B in our implementation) without packet-level fine-tuning, while also allowing users to substitute domain-specific or custom-trained models if desired.

The reasoning process follows a structured prompt (Fig. 4), guiding the model to parse the PIG, reconstruct event sequences, and evaluate their semantic plausibility. Detected anomalies may include off-path injection, PMTUD manipulation, NAT inconsistencies, or predictable UDP port allocations. With this global view, the LLM can attribute suspicious behavior to specific nodes or edges, generate concise explanations,

Input: Incoming network packets $P = \{p_1, p_2, ..., p_n\}$
Output: Dynamic Protocol Interaction Graph $G = (V, E)$

Initialization: Initialize empty graph G, BPF maps for nodes and edges, and per-packet call stack;
`RegisterProbes` (*functions* = {*ip_rcv, ipv6_rcv, tcp_v4_rcv, tcp_v6_rcv, tcp_ack, icmp_unreach, __ip_append_data, nf_nat_ipv4_in, ip_local_out*});
foreach *packet p in P* **do**
    trace ← `RunProbes`(*p*) ; `// eBPF triggers on function entry/return`
    foreach *function call f in trace* **do**
        node_id ← generate_node_id(*p*, *f*);
        `LogNode` (*node_id, context(p, f)*) ;
        caller_id ← `PopStack` ();
        if *caller_id* ≠ *null* **then**
            `LogEdge` (*caller_id* → *node_id, timestamp*) ;
        end
        `PushStack` (*node_id*) ;
    end
    `MergeTrace` (*trace, G*) ;
end
return G

ALGORITHM 1. Construction of PIG via eBPF.

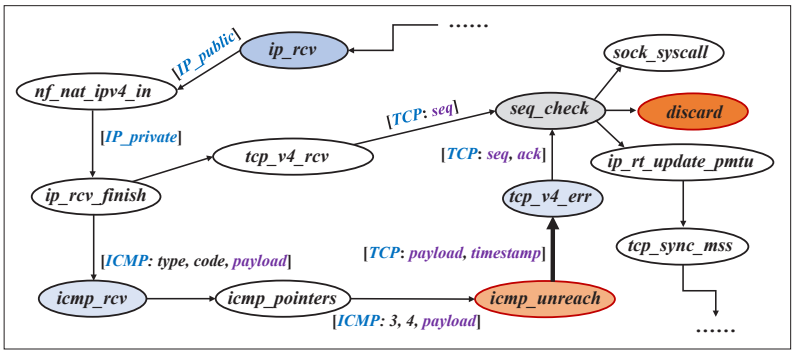

FIGURE 3. Partial protocol Interaction Graph (PIG) illustrating cross-layer packet processing in a PMTUD and NAT scenario.

[Task]
You are an expert in network protocol analysis and security.
Your task is to analyze a given Protocol Interaction Graph (PIG) and determine whether it indicates malicious or benign behavior.

[Input Format]
PIG = {
  Nodes: protocol events/calls (e.g., TCP_SYN, ICMP_Unreachable, NAT_Rewrite),
  Edges: causal dependencies between events,
  Context: call stack traces or packet metadata (if provided)
}

[Reasoning Steps]
1. Parse the sequence of nodes and edges in the PIG.
2. Identify anomalies, inconsistencies, or malicious exploits (e.g., off-path injection, PMTUD manipulation, NAT cross interactions, UDP source port prediction).
3. Provide reasoning: explain why the interaction is benign or malicious.
4. Conclude with a final classification: {Benign | Suspicious | Malicious}.

[Output Format]
- Classification and Confidence Score (%): <**Benign** / **Suspicious** / **Malicious**>
- Reasoning: concise explanation (3–5 sentences).
- Evidence: key suspicious nodes/edges in the PIG.

FIGURE 4. LLM prompt for identifying TCP/IP exploits within the Protocol Interaction Graph.

and output a classification (benign, suspicious, or malicious) with an associated confidence score that directly guides enforcement. By integrating PIG-based reasoning, `PacketScope` not only detects complex cross-layer exploits invisible to signature-based defenses but also pinpoints where in the TCP/IP stack an attack originates, what triggers it, and how it propagates.

For example, consider a scenario in which an off-path attacker on the Internet attempts to manipulate the path MTU of a NATed TCP host. The attacker impersonates an intermediate router along the host's path and sends forged ICMP "Fragmentation Needed" messages containing falsified TCP headers. The victim TCP host processes these messages through a sequence of functions captured in the Protocol Interaction Graph (PIG): `ip_rcv`, `nf_nat_ipv4_in`, `ip_rcv_finish`, `icmp_rcv`, `icmp_pointers`, `icmp_unreach`, `tcp_v4_err`, `seq_check`, and `discard`. Upon receipt, the IP layer forwards the ICMP messages through the NAT translation and delivers them to the ICMP handler. Based on the ICMP type and code, the message reaches the `icmp_unreach` function. Because the ICMP message carries an embedded TCP header, `tcp_v4_err` is invoked to verify whether the sequence number matches the expected value in the corresponding TCP flow's send window. If the sequence number is valid, the path MTU is updated; otherwise, the message is discarded. Throughout this process, the PIG records detailed interactions and causal dependencies between protocol layers.

Leveraging its ability to reason over extended context, the LLM identifies that the TCP sequence number in the embedded header conflicts with the actual flow state. Since the 32-bit sequence number is effectively random, the attacker's repeated ICMP messages are unlikely to match, resulting in numerous `icmp_unreach` calls and discarded messages. The model highlights the ICMP-TCP interactions as key evidence and constructs a reasoning chain explaining why this behavior is classified as malicious. It infers that a burst of ICMP "Fragmentation Needed" messages carrying incorrect TCP sequence numbers is abnormal and indicative of off-path PMTUD manipulation. This example illustrates the LLM's strength in correlating temporally and causally distant events, reasoning across multiple protocol layers, and uncovering attacks that might otherwise remain concealed. By combining PIG-based visibility with semantic reasoning, the system can detect subtle, cross-layer exploits with high precision.

It is worth noting that the LLM component in `PacketScope` introduces a distinct threat model, in which attackers may attempt PIG poisoning or prompt manipulation. Specifically, an attacker may craft benign-looking packet sequences to distort PIG construction and induce false positives or negatives, or attempt to influence LLM reasoning through prompt injection. To mitigate these risks, `PacketScope` incorporates multiple safeguards:

- Human-in-the-loop and cross-model validation, where LLM outputs are assigned confidence scores and only candidates exceeding a threshold (0.85 in our implementation) are compiled into eBPF rules after multi-LLM consensus and human review;
- Strict prompt sanitization using template-based, normalized, and whitelisted inputs, with no direct user- or network-controlled content;
- Shadow-mode deployment with automatic rollback, where newly generated rules are first deployed in observation-only mode with a bounded lifetime and promoted to enforcement only after passing safety and consistency checks.

Automatic rollback is triggered upon detecting abnormal traffic disruption or policy conflicts. Together, these mechanisms ensure that LLM-driven kernel enforcement remains robust.

In-Kernel Enforcement and Real-Time Blocking

In the final stage, `PacketScope` enforces the LLM-synthesized eBPF rules directly within the kernel. These rules are attached to strategic hook points, such as XDP (eXpress Data Path, eBPF-based high-performance network data path used to send and receive network packets at high rates) or TC (Traffic Control, user-space system administration utility program used to configure the Linux kernel packet scheduler), enabling early interception and high-performance packet handling. The LLM produces a compact policy specifying header- or state-based match predicates and corresponding actions, such as `drop`, `alert`, or `rate_limit`. The `PacketScope` compiler translates these predicates into eBPF programs: constant lookups (e.g., five-tuple identifiers, MTU thresholds) are stored in hash or LPM maps, while dynamic state (e.g., per-flow PMTU, token buckets) is maintained in LRU maps for $O(1)$ retrieval. Atomic updates ensure safe swapping of rules: each new eBPF program is verified, JIT-compiled, and installed with version tagging and TTL, enabling dynamic policy updates.

Consider the off-path PMTUD manipulation scenario described earlier, where forged ICMP "Fragmentation Needed" messages carry falsified TCP headers. `PacketScope` enforces kernel-level defenses using eBPF probes as shown in the following eBPF snippet.

```
SEC("xdp")
int xdp_pmtud_guard(struct xdp_md *ctx){
    void *data=(void*)(long)ctx->data,*end=(void*)(long)ctx
        ->data_end;
    // (1) Intercepting critical processing points.
    if((void*)(data+sizeof(struct ethhdr)+2*sizeof(struct
        iphdr)+sizeof(struct tcphdr))>end) return XDP_PASS;
    if(((struct ethhdr*)data)->h_proto!=__constant_htons(
        ETH_P_IP)) return XDP_PASS;
    // (2) Extracting embedded transport headers.
    struct icmphdr *icmp=(struct icmphdr*)((struct iphdr*)(
        data+sizeof(struct ethhdr))+1);
    struct iphdr *ip=(struct iphdr*)(icmp+1);
    struct tcphdr *tcp=(struct tcphdr*)(ip+1);
    if(((struct iphdr*)(data+sizeof(struct ethhdr)))->
        protocol!=IPPROTO_ICMP || icmp->type!=3 || icmp->
        code!=4 || ip->protocol!=IPPROTO_TCP) return
        XDP_PASS;
    // (3) Validating against live TCP flow state.
    struct tcp_meta *m=bpf_map_lookup_elem(&tcp_state,&
        make_flow_key(ip,tcp));
    if(!m) return XDP_PASS;
    return (bpf_ntohl(tcp->seq)<m->snd_una||bpf_ntohl(tcp->
        seq)>=m->snd_nxt)?XDP_DROP:XDP_PASS;
}
```

The enforcement proceeds in three stages. First, critical processing points are intercepted via XDP, parsing Ethernet and IP headers to filter IPv4 ICMP `type=3`, `code=4` packets. Second, embedded transport headers are extracted from the ICMP payload to reconstruct the TCP flow key, correlating the message with the corresponding live flow. Third, the embedded TCP sequence number is validated against the flow's send win-

No.	Attack Scenario	Exploit Path & Characteristics	PacketScope Defense	Total Tests	Snort (TP)	Kitsune (TP)	Flowrest (TP)	PacketScope
1	Off-path PMTUD Manipulation [12]	ICMP "Frag Needed" → TCP seq mismatch → forged ICMP flood	ICMP–TCP inconsistency detected and dropped by eBPF	100	0%	5%	2%	99% TP, 1% FN
2	ICMP Redirect Hijacking [1]	UDP probe → spoofed ICMP redirects	UDP–ICMP correlation and invalid redirects blocked	100	0%	8%	15%	98% TP, 2% FN
3	DoS via Forged TCP RST [6]	TCP probe → RST flood	PIG links probing and RSTs and flooding suppressed	100	45%	96%	94%	100% TP
4	DNS Cache Poisoning [2, 3]	UDP probe → fake DNS replies	UDP–DNS tracing and spoofed replies discarded	100	0%	12%	0%	100% TP
5	TCP Hijacking via IPID [1]	IPID leakage → TCP seq inference	IPID evolution traced and probing blocked in-kernel	100	0%	3%	0%	98%TP, 2% FN
6	NAT DoS [4]	RST flood → NAT mapping removal	TCP–NAT tracing and inconsistent RSTs blocked	100	10%	65%	58%	100% TP
7	Benign Traffic~	Normal HTTP/FTP/Video traffic	Normal forwarding	2,000	N/A	N/A	N/A	3/2000 = 0.15% FP

TABLE 1. Defense results of PacketScope against representative TCP/IP exploits.

dow stored in the `tcp_state` map. Messages with invalid sequence numbers are dropped immediately, while valid packets proceed to the kernel TCP handler.

To prevent incorrect eBPF rules from disrupting normal connectivity, `PacketScope` adopts a multi-stage safety and validation pipeline. Newly generated rules are first deployed in *shadow mode*, where matches are logged without enforcement. These logs are compared against baseline traffic statistics, and rules are promoted to enforcement only if the false positive rate remains below a predefined threshold. In addition, rules affecting control-plane or management traffic (e.g., BGP and SSH) are subject to human-in-the-loop validation. When multiple LLM instances are available, `PacketScope` further applies consensus-based validation, accepting rules only upon reaching a predefined agreement threshold. Each enforced rule is continuously monitored and bounded by a TTL, with automatic deactivation upon abnormal behavior or policy conflicts. A versioned rollback mechanism enables rapid recovery by restoring the last-known-safe configuration, including disabling all LLM-generated rules when necessary. In practice, `PacketScope` also caches validated LLM-generated eBPF rules. When a new flow's PIG matches a previously identified and verified attack, the corresponding cached rule is applied immediately without re-invoking the LLM, reducing response and inference overhead. Our prototype implements such a rule cache to support efficient reuse.

Implementation and Evaluation

Implementation

We implement a prototype of `PacketScope` on Ubuntu 24.04 LTS with Linux kernel 6.8. The in-kernel enforcement engine is developed using the BCC (BPF Compiler Collection) framework, with performance-critical modules selectively migrated to `libbpf` for efficiency. eBPF programs are attached to kernel functions via kprobes and tracepoints, targeting critical paths such as `ip_rcv`, `icmp_rcv`, and `tcp_v4_err`. For data-plane enforcement, packet filters are deployed at both the XDP and TC ingress hooks, enabling early packet dropping at near line-rate performance. To construct protocol interaction graphs, we implement a custom tracer that hooks into both transport and network layers. Semantic analysis is performed by a general-purpose pre-trained LLM (DeepSeek-67B in our implementation) running on NVIDIA A100 GPUs, without packet-level fine-tuning, though custom models are supported.

To handle long traces, PIGs are encoded in a hybrid JSON format combining linearized call traces with compressed adjacency lists. Prompts integrate structured indicators (e.g., inconsistent PMTU updates) with natural-language cues. The model generates reasoning traces and candidate enforcement predicates, which are validated and compiled into eBPF bytecode via clang/LLVM (v15.0) before being injected into the kernel. Enforcement follows a two-phase workflow: rules are first deployed in shadow mode for observation and promoted atomically to commit mode upon validation. `PacketScope` is designed for security-critical endpoints and network gateways (e.g., cloud servers and enterprise firewalls), where GPU resources or centralized LLM inference services are available. The LLM component operates asynchronously and is decoupled from the data plane, enabling scalable semantic analysis without disrupting normal packet forwarding.

Effectiveness Evaluations

As summarized in Table 1, we deploy representative TCP/IP attack scenarios exploiting semantic inconsistencies across protocol interactions in our testbed and collect execution traces to assess the effectiveness of `PacketScope`. Attacks are launched using custom scripts and modified versions of open-source tools (e.g., scapy) to generate protocol-compliant but semantically malicious packets. `PacketScope` outperforms baseline systems — Snort [13], Kitsune [14], and Flowrest [15] — achieving an average True Positive (TP) rate above 98%. For example, in off-path PMTUD attacks, forged ICMP "Fragmentation Needed" messages with inconsistent TCP state are identified and dropped. Across 100 trials, 99 are successfully detected, yielding a 99% TP rate and 1% False Negative (FN) rate. This advantage stems from its ability to expose semantic inconsistencies by constructing PIGs and applying LLM reasoning.

We further assess False Positives (FP) of

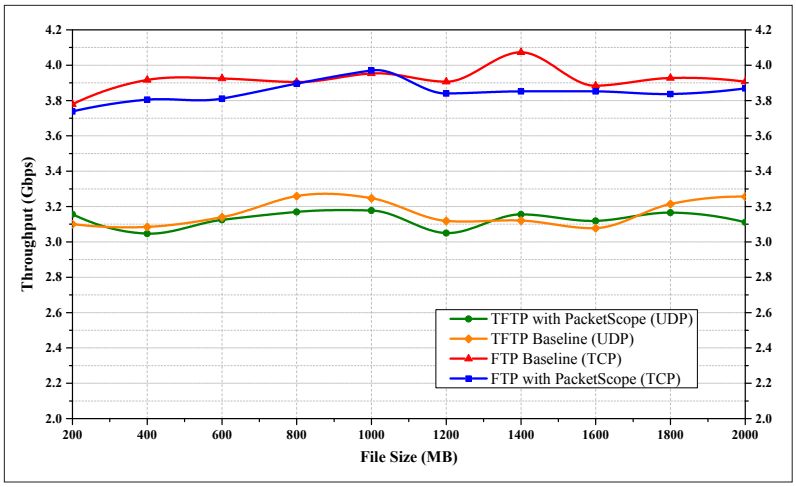

FIGURE 5. Throughput comparison of FTP and TFTP workloads with and without `PacketScope` enabled.

`PacketScope` using 2,000 benign flows (HTTP, FTP, and video) with injected jitter and packet reordering. Even under these challenging conditions, `PacketScope` achieves a low average false positive rate of 0.15%. The false positives result from transient network artifacts, such as out-of-order ICMP messages or TCP retransmissions. To avoid disrupting legitimate traffic, `PacketScope` employs a shadow mode in which provisional rules are logged and validated against subsequent traffic behavior, ensuring robustness against transient network instability.

Performance Evaluations

We evaluate the performance impact introduced by `PacketScope` in a client–server setup running FTP (TCP-based) and TFTP (UDP-based) workloads. In the baseline configuration, `PacketScope` is disabled; in the experimental configuration, it is enabled with eBPF probes attached to relevant kernel functions and LLM-generated enforcement rules activated. As shown in Fig. 5, file download throughput measured across file sizes ranging from 200 MB to 2000 MB shows that `PacketScope` incurs on average less than 2.3% performance loss for FTP and less than 1.6% for TFTP. LLM inference is carried out asynchronously and out-of-band on the dedicated GPU machine. Therefore, the throughput measurements primarily reflect the cost of in-kernel enforcement and do not account for the LLM inference overhead. Beyond throughput, the average system overhead remains modest: CPU usage increases by 1.61% at 100 Mb/s; each per-flow PIG state averages 1800 B; eBPF maps consume less than 20 MB of memory with 10k concurrent flows; and 67B-LLM inference typically requires 40 GB of GPU memory and 12 s on an A100. The end-to-end rule-deployment pipeline averages below 9 s (PIG generation 2 s, LLM analysis 6 s, compilation 20 ms, kernel loading 10 ms), while asynchronous LLM processing leaves packet forwarding unaffected. Note that for high-load servers, full PIG construction may be impractical; `PacketScope` therefore supports optional flow sampling (i.e., by selected IP address pairs) to trade detection coverage for performance, which we identify as a limitation.

Conclusion and Future Work

We present `PacketScope`, which combines kernel-level observability with AI-driven semantic analysis to mitigate subtle TCP/IP exploits. By reconstructing protocol interaction graphs from TCP/IP stack and applying LLM-guided reasoning, `PacketScope` translates rich behavioral telemetry into precise eBPF enforcement policies for in-stack blocking. A Linux prototype integrating eBPF and DeepSeek demonstrates effective detection of diverse exploits with acceptable performance overhead. In the future, we plan to extend `PacketScope` into a general-purpose in-kernel enforcement point that supports runtime security policies beyond networking, such as system calls, file operations, and application-level behaviors. By incorporating eBPF-observed system dynamics outside the TCP/IP stack into long-context LLM analysis, `PacketScope` aims to detect more complex, multi-stage, and previously concealed attacks.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants 62425201, 62132011, and 62221003, and by the Beijing–Tianjin–Hebei Natural Science Foundation Cooperation Project under Grant 25JJJJC0003.

References

[1] X. Feng *et al.*, "Exploiting Cross-Layer Vulnerabilities: Off-Path Attacks on the TCP/IP Protocol Suite," *Communications of the ACM*, vol. 68, no. 3, 2025, pp. 48–59.
[2] K. Man *et al.*, "DNS Cache Poisoning Attack Reloaded: Revolutions with Side Channels," *Proc. 2020 ACM SIGSAC Conf. Computer and Commun. Security*, 2020, pp. 1337–50.
[3] K. Man, X. Zhou, and Z. Qian, "DNS Cache Poisoning Attack: Resurrections with Side Channels," *Proc. 2021 ACM SIGSAC Conf. Computer and Communications Security*, 2021, pp. 3400–14.
[4] X. Feng *et al.*, "REDAN: An Empirical Study on Remote DoS Attacks Against NAT Networks," *Network and Distributed System Security Symp. (NDSS)*, 2025.
[5] L. Torvalds, "ipv4: TCP: Send Zero IPID in Synack Messages;" https://github.com/torvalds/linux/commit/970a5a3ea86da637471d3cd04d513a0755aba4bf, accessed on: Dec. 5, 2025.
[6] Y. Cao *et al.*, "Off-Path TCP Exploits: Global Rate Limit Considered Dangerous," *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 209–25.
[7] E. Dumazet, "[Patch Net] TCP: Make Challenge Acks Less Predictable;" https://www.mail-archive.com/netdev@vger.kernel.org/msg118677.html, accessed on: December 5, 2025.
[8] T. Cui *et al.*, "TrafficLLM: Enhancing Large Language Models for Network Traffic Analysis with Generic Traffic Representation," accessed on: Dec. 5, 2025; https://arxiv.org/abs/2504.04222
[9] A. Klein, "Cross Layer Attacks and How to Use Them (for DNS Cache Poisoning, Device Tracking and More)," *2021 IEEE Symp. Security and Privacy (S&P)*, 2021, pp. 1179–96.
[10] V. Paxson, "Zeek," https://zeek.org/, Accessed on: December 5, 2025.
[11] —, "Bro: A System for Detecting Network Intruders in Real-time," *7th USENIX Security Symp. (USENIX Security 98)*, 1998.
[12] X. Feng *et al.*, "Off-Path TCP Exploits: Pmtud Breaks Tcp Connection Isolation in IP Address Sharing Scenarios," *Proc. 2025 ACM SIGSAC Conf. Computer and Communications Security (CCS)*, 2025, pp. 4574–87.
[13] M. Roesch, "Snort," https://www.snort.org/, Accessed on: Dec. 5, 2025.
[14] Y. Mirsky *et al.*, "Kitsune: an Ensemble of Autoencoders for Online Network Intrusion Detection," *Network and Distributed System Security Symp. (NDSS)*, 2018.
[15] A. T.-J. Akem, M. Gucciardo, and M. Fiore, "Flowrest: Practical Flow-Level Inference in Programmable Switches with Random Forests," *IEEE Conf. Computer Communications (IEEE INFOCOM)*, 2023.

Biographies

Xuewei Feng (fengxw06@126.com) received his Ph.D. degree from Tsinghua University, where he is currently a Research Scientist. His research interests include network security and software vulnerability detection.

Qixun Tang received the B.E. degree from Tsinghua University. He is currently pursuing the Ph.D. degree at the Institute of Software, Chinese Academy of Sciences, and Zhongguancun Laboratory. His research interests include network and system security.

Min Li received the M.E. degree from the Institute of Software, Chinese Academy of Sciences. She is currently an engineer at Zhongguancun Laboratory. Her research interests include software vulnerability detection and program analysis.

Yuxiang Yang received the B.E. degree from Tsinghua University. He is currently pursuing the Ph.D. degree at Tsinghua University. His research interests include network security and machine learning.

Zhaoxi Li received the B.E. degree from Beijing Jiaotong University. He is currently pursuing the Ph.D. degree at Tsinghua University. His research interests include protocol vulnerability detection and machine learning.

Xingxiang Zhan received the M.E. degree from Zhejiang University. He is currently an engineer at Zhongguancun Laboratory. His research interests include network security and IoT vulnerability detection.

Yi He received the Ph.D. degree from Tsinghua University. He is currently an Assistant Professor at Wuhan University. His research interests include system security and program analysis.

Chi Chen received the B.E. degree from Shanghai University. He is currently a Senior Technical Expert at Meituan Corporation. His research interests include network security and system operations.

Qi Li received the Ph.D. degree from Tsinghua University, where he is currently an Associate Professor with the Institute for Network Sciences and Cyberspace. His research interests include Internet and cloud security, IoT security, and AI security. He serves on the Editorial Boards of *IEEE TDSC* and *ACM TOPS*.

Ke Xu [F] (xuke@tsinghua.edu.cn) received his Ph.D. degree from Tsinghua University, where he is currently a Full Professor. He has published over 200 technical papers in the areas of next-generation Internet and network security. He is a member of ACM and has served as a guest editor for several special issues of IEEE and Springer journals.