Exploring and Analyzing Cross Layer DoS Attack Against UDP-based Services on Linux

Dashuai Wu
wds24@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Yunyi Zhang
zhangyyzyy@nudt.edu.cn
Tsinghua University
Beijing, China
National University of Defense Technology
Changsha, China

Baojun Liu
lbj@tsinghua.edu.cn
Tsinghua University
Beijing, China

Xiang Li
lixiang@nankai.edu.cn
Nankai Univeristy
Tianjin, China

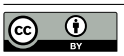
Eihal Alowaisheq
ealowaisheq@ksu.edu.sa
King Saud University
Riyadh, Saudi Arabia

Haixin Duan
duanhx@tsinghua.edu.cn
Tsinghua University
Beijing, China

Abstract

The layered architecture of the TCP/IP protocol stack enables protocol layers to be implemented independently and flexibly. However, this layered design introduces potential security risks when shared resources are not properly managed between different layers.

This paper investigates a neglected cross-layer shared resource risk, termed `SocketFilled`, which exploits the insecure usage of the UDP send buffer at the transport layer by the link layer, resulting in the interruption of response packets from the upper application layer. To explore the root causes of cross-layer DoS vulnerabilities resulting from the implementation of the TCP/IP protocol stack, we systematically analyzed the protocol standards of address resolution and reviewed the implementation in mainstream open-source operating systems. Moreover, we conducted a comprehensive experimental evaluation of mainstream operating systems (e.g., Linux and FreeBSD) and UDP services (e.g., DNS and QUIC). The experimental results show that the latest version of Linux and UDP service software (e.g., BIND9, PowerDNS, and Nginx) are affected, causing significant packet loss and even complete service interruption. Then, we estimated the impact range of `SocketFilled` in the wild and demonstrated that 17.3% of open resolvers, 54.3% of authoritative servers of the Tranco Top 100K domains, and 3.8% of these well-known domains' HTTP/3 servers are potentially affected, including Bing, Amazon, and Shopee, after excluding the influence of cloud servers. We have conducted responsible disclosure by reporting the vulnerability to the Linux community. Our research highlights the effectiveness of cross-layer mechanisms in DoS attacks and calls for heightened attention to the layered complexity of protocol stack implementations within the security community.



CCS '25, Taipei, China

ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3744878

CCS Concepts

• **Networks** → **Cross-layer protocols**; • **Security and privacy** → **Denial-of-service attacks**.

Keywords

Cross-Layer Attack, Denial-of-Service Attack, Linux Neighbor Subsystem, UDP

ACM Reference Format:
Dashuai Wu, Yunyi Zhang, Baojun Liu, Xiang Li, Eihal Alowaisheq, and Haixin Duan. 2025. Exploring and Analyzing Cross Layer DoS Attack Against UDP-based Services on Linux. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, China.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3719027.3744878

1 Introduction

Denial of Service (DoS) attacks, which are a form of network attack that compromises service availability, prevent legitimate users from accessing target services by exhausting system resources (such as network bandwidth and computing resources) or exploiting vulnerabilities to disrupt services. In recent years, numerous large-scale Distributed Denial of Service (DDoS) attacks have been reported. For instance, the 2016 attack against Dyn DNS disrupted multiple major Internet services including Twitter and Netflix, causing significant economic losses [14]. Akamai's latest State of the Internet report shows a fivefold increase in the number of Layer 7 DDoS attacks targeting Asia Pacific and Japan from January 2023 to June 2024 [1].

Flood attacks that consume the victim's resources with a large amount of traffic are common denial-of-service attacks, such as HTTP Flood [3, 15] and UDP Flood [9, 22]. Botnet services that can be rented cheaply on the darknet enable adversaries to carry out such attacks without comprehensive technical knowledge [2]. To guard against these attacks, both academia and industry have proposed numerous defense methods, such as traffic filtering, rate limiting, and anomaly detection techniques [8, 31]. Another category of DoS attacks exploits service or system vulnerabilities to directly compromise the availability of services, such as Keytrap [19], TuDoor [26] and Slow Post DDoS Attack [35]. These attacks

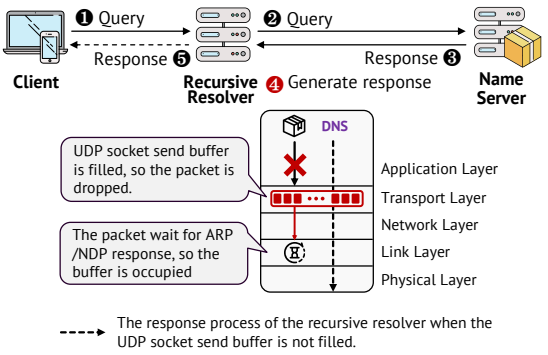


Figure 1: An example of the SocketFilled attack. When the UDP socket buffer is filled by the attacker, the DNS resolver cannot generate valid responses, leading to the failure of the client's domain name query.

do not have distinct traffic features, making them more challenging to defend against.

Research Gap. Previous studies have leveraged different services on the Internet to construct new types of DoS attacks, such as DNS [12, 25], CDN [24, 28], etc. They exploit service vulnerabilities to orchestrate large-scale attack traffic [26] or directly deplete the computing resources of the target server [19]. In addition, recent studies revealed that cross-layer side channels can affect the security of upper-layer applications, such as DNS cache poisoning [30] and TCP injection [6]. Moreover, the community mailing list once discussed a kernel UDP behavior with missing destination [29], which can potentially degrade the performance of the upper layer application. However, *no prior work has systematically analyzed how cross-layer implementation flaws in protocol stacks affect upper-layer application availability or evaluated their actual impacts in the wild.*

Cross-layer Shared Resource Risks. The UDP socket's send buffer is shared between the link layer and transport layer in the protocol stack. Link layer address resolution mechanisms of the Linux kernel address resolution module (Linux neighbor subsystem), including ARP for IPv4 and NDP for IPv6, may unsafely occupy the transmission buffer space resources of UDP sockets at the transport layer. The Linux neighbor subsystem uses queue-based pending packet management and a fixed timeout strategy. When encountering an address that cannot be resolved (regardless of whether it has been queried before), it will keep waiting until the timeout. Exploiting these unresolvable neighbors to fill the UDP socket's send buffer, attackers are able to impact the availability of upper-layer applications, achieving a cross-layer DoS attack.

SocketFilled Attack. Specifically, we observed that the Linux neighbor subsystem does not maintain a history of queries and has a small socket send buffer. This suggests that attackers can easily use invalid queries (i.e., *unresolvable neighbors*) to fill up the buffer. We refer to this type of attack as the `SocketFilled` attack. The `SocketFilled` attack exploits the unsafe address resolution timeout behavior in Linux systems. During address resolution, the Linux neighbor subsystem holds outgoing packets until a timeout occurs, occupying space in the socket's send buffer, as shown in Figure 1. By exploiting this mechanism, attackers can force the socket's send buffer to be filled with packets waiting for unresolvable address resolution, causing response packets to legitimate users to be dropped due to insufficient buffer space (see Section 3).

Our Study. In our research, we investigated the implementation of the operating system network stack that enables the `SocketFilled` attack. We conducted a thorough analysis of relevant RFC specifications and performed a comparative study of protocol stack implementations across two widely deployed open-source operating systems (Linux and FreeBSD), which provided a comprehensive understanding of the underlying mechanisms that enable the SocketFilled attack. Our investigation reveals that the vulnerability originates from previously unidentified hazardous interactions between the Linux neighbor subsystem and its unique transport layer implementation. This vulnerability makes all UDP-based services on Linux systems susceptible to potential attacks. Furthermore, we identified several critical kernel parameters associated with this vulnerability and demonstrated how their default configurations contribute to the system's susceptibility (see Section 4).

We demonstrated that the `SocketFilled` attack is completely realistic and has a broad impact. We conducted tests on 4 mainstream operating systems, including Windows, Linux, macOS, and FreeBSD. Specifically, we deployed the simple UDP Echo service on the latest versions of the operating systems to verify their vulnerability. The results showed that the latest version of Linux is affected by `SocketFilled`. Moreover, we evaluated 9 software applications of two mainstream UDP protocols (DNS and QUIC) on the latest Linux systems (see Section 5). Our results demonstrate that 8 tested applications are vulnerable to the `SocketFilled` attack, including BIND9, PowerDNS, and Nginx. With as few as hundreds of packets, an attacker can induce significant packet loss or even render the targeted services completely unresponsive. The `SocketFilled` attack requires the attacker to spoof the source address, and assumes that the target network does not have a source address verification mechanism in place, which is entirely feasible in real-world networks [5].

Furthermore, we estimated the potential impact of the SocketFilled attack in real-world scenarios by conducting measurements on open DNS resolvers and Tranco Top 100K domains [23], focusing on their nameservers and QUIC services. We discovered that network configurations of cloud service platforms, such as spoofing responses to unreachable IP addresses, can mitigate the attack. Therefore, our main testing targets are services that are not deployed on cloud platforms. Specifically, we utilized traceroute to estimate the number of unresolvable neighbor IP addresses for the target service, thereby determining whether it is a potential victim. Our measurement results indicate that the `SocketFilled` attack has a considerable number of potentially affected targets. (see Section 6). We identified that 17.3% of open resolvers, 54.3% of authoritative nameservers of the Tranco Top 100K domains, and 3.8% of these popular domains' HTTP/3 servers are potentially affected, including Bing, Amazon, and Shopee.

Mitigation and Disclosure. To defend against `SocketFilled`, we propose three countermeasures for different stakeholders based on our analysis of Linux mechanisms and experimental results. Moreover, we have reported the vulnerabilities to the Linux community and demonstrated the concrete and actual impact of the attack on

application-layer software, calling on the community to increase their attention to this vulnerability (see Section 7).

Our research demonstrates the significant threats posed by cross-layer attacks. Particularly in complex multi-layer TCP/IP protocol stacks, security vulnerabilities introduced by inter-layer interactions and shared resource usage are difficult to detect. The long-standing dependence of upper-layer applications on lower-layer mechanisms makes these vulnerabilities even more challenging to remediate. Defense measures implemented at the application layer can rarely provide complete protection against lower-layer threats, while attackers can exploit such dependencies to easily compromise numerous upper-layer applications by leveraging vulnerabilities in the lower layers.

Contributions. We make the following contributions:

- *Systematic analysis of cross-layer DoS attack.* We investigate the neglected and realistic cross-layer DoS attack that exploits vulnerabilities in the interactions across layers within the Linux kernel network stack.
- *Comprehensive evaluation of realistic attack.* We conducted a systematic evaluation of 4 mainstream operating systems and 9 software implementations of two popular UDP protocols on the latest Linux systems and estimated the potential impact in the wild.
- *Mitigation measures and responsible disclosure.* We propose multiple mitigation measures and conduct responsible disclosure to the Linux kernel community.

2 Background and Related Work

In this section, we first present an overview of DoS attacks, followed by the technical background on address resolution mechanisms and Linux network stack implementation details that are fundamental to our work. Finally, we summarize the related work pertaining to our research topic.

2.1 DoS Attacks

A DoS attack is a prevalent form of attack in which an attacker deliberately disrupts the availability of services hosted by a network-connected system, rendering the targeted machine or network resources inaccessible to legitimate users, either temporarily or indefinitely. DoS attacks can be broadly classified into distributed denial-of-service (DDoS) attacks and conventional denial-of-service (CDoS) attacks. DDoS attacks typically leverage large-scale botnets to achieve service disruption through resource exhaustion, while CDoS attacks exploit specific vulnerabilities in protocols or implementations to achieve targeted denial effects.

DoS attacks have evolved significantly over the past decade, from simple volumetric attacks to more sophisticated multi-vector approaches. Classical DoS methods include network-layer attacks such as ICMP floods, UDP floods, and TCP SYN floods, which aim to exhaust network bandwidth or connection resources. Recent variants focus on application-layer attacks, particularly HTTP floods that mimic legitimate traffic patterns. Amplification attacks represent another prevalent category, where attackers abuse infrastructure services (e.g., DNS, CDN) as reflectors to multiply attack traffic volume. Moreover, different forms of DDoS attacks are being actively exploited. Cloudflare's DDoS Threat Report for Q2 2024 [7] indicates a 20% year-over-year increase in DDoS attacks in 2024, underscoring the severity of DDoS threats.

2.2 Address Resolution and Linux Socket Send Buffer

Address resolution refers to the process of converting network layer IP addresses to data link layer MAC addresses in the TCP/IP protocol stack. Hosts need to determine the MAC address of the next hop to correctly forward data packets. Having hosts perform address resolution rather than relying entirely on routers helps reduce unnecessary routing query overhead, although this may be negligible in modern networks (see Section 6.1). In IPv4 networks, this functionality is accomplished through ARP (Address Resolution Protocol) [39], while in IPv6 networks, it is handled by NDP (Neighbor Discovery Protocol) [42] which is integrated into ICMPv6 [18]. Due to its location at the lower layers of the network protocol stack and the near impossibility of remote exploitation, address resolution has received minimal attention from security researchers.

In Linux systems, applications typically send network packets through socket system calls. After an application transmits data to the kernel via system calls, the kernel allocates an `sk_buff` (Linux's packet storage structure) in the sending buffer of the corresponding socket. The application layer content is then copied into this structure, followed by layer-by-layer processing and protocol header population within the kernel network stack. The `sk_buff` space is released either upon successful transmission or when an error occurs. If the remaining space in the socket's sending buffer is insufficient to accommodate the pending data, the corresponding system call discards the packet and returns an error message to the upper-layer application. Applications can independently retry the transmission of failed packets.

Of particular relevance to our work is the interaction between Linux socket send buffer management and its address resolution operations. The Linux kernel's implementation creates implicit dependencies between these components, potentially enabling cross-layer effects that traditional security analyses might overlook. This observation forms a crucial foundation for our attack methodology, as detailed in subsequent sections.

2.3 Related Work

DoS Attacks. Current research on DoS attacks primarily focuses on two aspects: the discovery of novel amplification mechanisms and DoS defense strategies. In terms of amplification mechanisms, Lin et al. [28] proposed leveraging CDN performance-oriented origin-fetching policies to trigger amplification attacks against origin servers, while Xu et al. [47] demonstrated the possibility of utilizing DNS resolvers as efficient amplifiers through combination techniques. Regarding defense strategies, approaches such as multi-hop traffic dispersion [27] and reactive BGP routing [43] have been proposed to counter DDoS attacks.

In the realm of CDoS attacks, research has primarily focused on identifying and exploiting vulnerabilities in specific protocols and scenarios. Xiong et al. [46] demonstrated the serverless function DoS attack through IP reputation poisoning. Mirkin et al. [33]

introduced blockchain-targeted DoS attacks exploiting proof-of-work mechanisms, and Nguyen et al. [36] exploited web cache poisoning techniques. These targeted attacks require a sophisticated understanding of system mechanics and often achieve denial effects through indirect means rather than direct resource exhaustion.

Cross-Layer Attacks. Cross-layer attacks represent a sophisticated category of security threats that exploit vulnerabilities spanning multiple layers of the network protocol stack. These vulnerabilities are often highly stealthy, making them difficult to detect, and their impact can be profoundly severe. These attacks can manifest in three primary forms: (1) vulnerabilities in one layer affecting other layers, (2) insecure interactions between protocol layers, and (3) side-channel attacks leveraging lower-layer information leakage.

Preliminary investigations into cross-layer attacks have been initiated in several studies. Feng et al. [16] demonstrated TCP connection hijacking by exploiting Linux's IPID allocation mechanisms. Klein [21] showcased how compromised pseudo-random number generators could affect protocol security across multiple layers. Feng et al. [17] achieved man-in-the-middle attacks in WiFi networks through ICMP redirect exploitation, while Man et al. [30] utilized ICMP rate limiting as a side channel for DNS cache poisoning. Notably, these attacks predominantly leverage cross-layer vulnerabilities to attack clients through data leakage or identity spoofing, resulting in hijacking or poisoning effects. However, limited attention has been paid to the potential of cross-layer attacks in DoS attacks. In particular, *it remains unclear whether flaws in the cross-layer implementation of protocols could directly impact the availability of upper-layer applications.*

3 Attack Overview

In this paper, we present a novel cross-layer DoS attack called SocketFilled. This attack exploits unsafe interactions between the neighbor subsystem and the transport layer implementation in the Linux kernel network stack to achieve efficient denial-of-service attacks against UDP-based services. With this attack, we highlight that the layered design principle of the TCP/IP stack increases the independence of each layer while simultaneously posing challenges for the design of interactions between different layers and that interaction flaws may introduce new risks.

In this section, we first introduce the threat model of SocketFilled, followed by the workflow for conducting the attack.

3.1 Threat Model

Figure 2 illustrates the threat model of the SocketFilled attack, which aims to prevent the service from responding to legitimate requests by exploiting the socket send buffer and address resolution mechanism in cross-layer communication. The model consists of four components: 1) a target server (which may be a DNS authoritative server, DNS recursive server, QUIC/HTTP3 server, or other UDP-based service host), 2) unresolvable neighbor IP addresses, which belong to the same network as the target host, 3) legitimate users of the target host (which may be DNS recursive servers, DNS clients, QUIC clients, or other clients), and 4) an off-path attacker. The off-path attacker aims to prevent the service from responding to legitimate requests. By forging request packets, the attacker can induce the target server to send packets to unresolvable IP addresses, which triggers pointless and pending address resolution requests (i.e., ARP or NDP requests). Therefore, the socket send buffer is filled up, causing normal response packets to be dropped. To execute the SocketFilled attack successfully, the following requirements must be met:

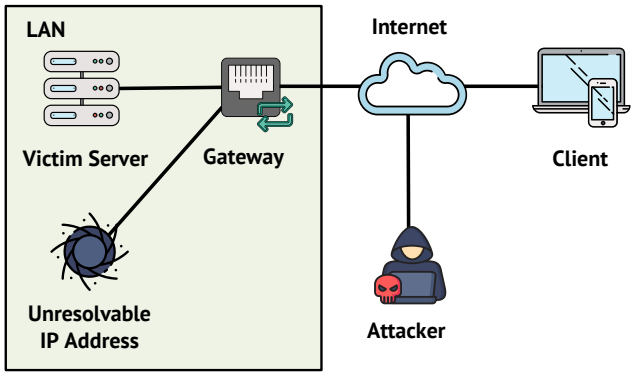


Figure 2: Threat Model of the SocketFilled Attack.

IP Spoofing. The attacker is capable of sending packets with unresolvable neighbor IP addresses. According to the report from CAIDA [5] in December 2024 , approximately one-fifth of IPv4 and one-quarter of IPv6 Autonomous Systems (ASes) permit IP spoofing, enabling attackers to routinely perform source address spoofing by renting bulletproof hosts in these ASes. Furthermore, research [10] indicates that approximately 70% of ASes do not implement ingress filtering, which exacerbates the severity of IP address spoofing.

Vulnerable Target. The victim is a UDP-based request-response service running on Linux systems. Thus, attackers can trigger the victim to send responses to unreachable IP addresses by faking requests. Linux is the most widely deployed operating system in server environments, and several UDP-based application layer protocols, such as DNS and QUIC, are extensively used across the Internet. Additionally, the victim must have vulnerable system parameters (see Section 4).

Unresolvable Neighbor IP Addresses. The victim must direct packet transmission by resolving the neighbor host's MAC address through ARP or NDP, rather than routing packets through a gateway. As a result, unresolvable neighbor IP addresses (either unassigned or non-responsive to address resolution requests) will exist in the victim's network.

For IPv4 networks, this depends on the subnet assignment and the IP address utilization within the subnet. For IPv6, since RFC mandates that the most specific prefix assigned to end hosts is /64, the existence of sufficient unresolvable neighbor IP addresses is evident. In addition, special measures implemented by cloud service providers may also impact this condition (see Section 6).

Attackers can infer potential unresolvable neighbor IP addresses by analyzing the victim's IP address structure and performing preliminary verification through ICMP echo requests, as demonstrated in Section 6.

3.2 Attack Workflow

Figure 3 illustrates the key steps of the SocketFilled attack. Initially, clients can request services and receive responses from the server. The server can be any Linux UDP service, such as DNS authoritative servers, DNS resolvers, or HTTP/3 servers. An off-path attacker aims to prevent clients from receiving responses from the server.

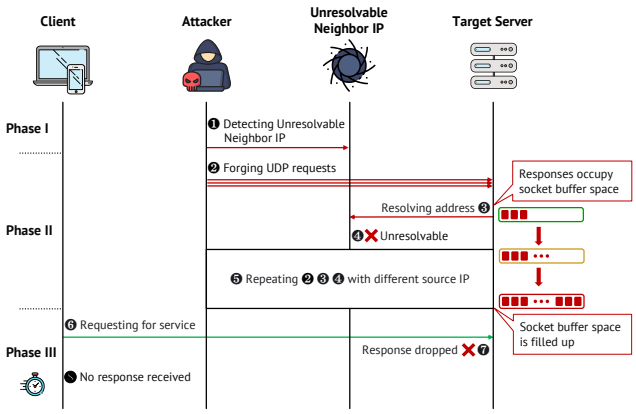


Figure 3: Overview of the SocketFilled Attack.

The attack consists of eight steps that can be broken down into three main phases:

Phase I: Preparation. The preparation phase begins with the attacker's reconnaissance activities. Through careful analysis of the victim's address structure and the strategic deployment of ICMP echo requests, the attacker identifies a sufficient quantity of unresolvable neighbor IP addresses (step ❶). Our experimental results show that attackers only need a small number of IP addresses to meet the attack requirements, as elaborated in Section 4.

Phase II: Attack Initialization. The attacker initiates the attack by transmitting forged request packets to the victim, with spoofed source addresses corresponding to the previously identified unresolvable neighbor addresses (step ❷). Upon receiving these requests, the victim attempts to resolve these unresolvable neighbor addresses through the address resolution mechanism (step ❸), such as ARP in IPv4 and NDP in IPv6. As expected, these resolution attempts fail due to the non-existence or non-responsiveness of the spoofed neighbor IP addresses (step ❹). However, the response packets constructed by the victim remain in memory, continuously occupying the socket's send buffer space. The attacker then repeatedly executes step ❷-❹, ultimately resulting in the complete occupation of the victim's socket buffer space (step ❺).

Phase III: Service Disruption. During the interval between complete buffer occupation and the neighbor subsystem's timeout-triggered queue clearance, legitimate clients attempt to access the victim's services (step ❻). When the victim attempts to generate responses, these response packets are dropped due to the lack of socket send buffer space (step ❼). Consequently, clients experience a service disruption as they fail to receive any responses from the victim (step ❽).

This attack workflow demonstrates *how a relatively simple sequence of steps can lead to significant service disruption* by exploiting socket buffer management mechanisms in cross-layer interaction.

4 Systematic Analysis of Address Resolution and UDP Packet Management Mechanism

In local area networks (LANs), which constitute the fundamental branches of the Internet, the address resolution mechanism maps network-layer addresses to link-layer addresses. This mechanism not only ensures normal communication within LANs but also eliminates additional routing through routers for intra-LAN data packets. In Ethernet, the most widely deployed LAN protocol, the common implementations are ARP for IPv4 and NDP for IPv6.

To systematically analyze how the address resolution mechanism affects upper-layer applications, we first review neighbor discovery-related RFCs in this section. The analysis focuses on the specified behaviors and pending packet management mechanisms in ARP and NDP when neighbors do not exist, which is particularly relevant to `SocketFilled` attacks. We then conduct a detailed source code analysis of the corresponding network protocol stack components in two major open-source operating systems, Linux and FreeBSD, examining their implementation of these specifications and their management approaches for UDP pending packets. Through comparative analysis, we identify an unsafe interaction in Linux between link-layer neighbor discovery and transport-layer UDP pending packet management. Finally, we analyze several parameters closely related to this vulnerability in Linux systems and highlight the security implications of their default values.

4.1 Address Resolution Specifications

We first conducted a comprehensive analysis of the three core RFCs of the address resolution mechanism, namely, RFC 826 [39], RFC 1122 [4] and RFC 4861 [42], to confirm the requirements of the protocol standards and provide foundational insights for further code analysis.

Address Resolution Protocol. The ARP specification [39] defines a fundamental mechanism for network address resolution, which entails the maintenance of a translation table. A table entry is represented as follows:

$$< protocol\_types, protocol\_addresses, hardware\_addresses >$$

When packet transmission is initiated by upper layers, the address resolution module looks up the table to find the target host's MAC address. In cases where the required mapping is absent from the table, the address resolution module initiates an ARP resolution process by broadcasting an ARP request and then inserts the new mapping into the table.

We observed that the specification does not adequately discuss *how unresponsive ARP requests should be handled, particularly under anomalous conditions*, although it requires the link layer to maintain at least one unsent data packet for each destination during address resolution [4]. This significant omission can be attributed to the historical context in which ARP was developed: during the Internet's early stages, when network environments were relatively simple and trustworthy, and communication failures with hosts were considered exceptional cases. However, in today's sophisticated and

often hostile network environments, as our subsequent analysis will demonstrate, this specification gap has resulted in divergent implementations across different systems, potentially introducing significant security vulnerabilities.

Neighbor Discovery Protocol. In contrast to ARP, the NDP specification [42]was developed with a comprehensive understanding of modern network complexity and security challenges, providing detailed guidelines for handling various network scenarios. Note that we focus on the protocol's sophisticated approach to managing outgoing packets and state transitions during the address resolution process, which is particularly relevant to the SocketFilled attack.

The specification mandates a well-defined system where senders must maintain separate queues for outgoing packets for each neighbor. To prevent resource exhaustion, the specification recommends implementing modest queue size limits and employing a first-in-first-out replacement strategy when queue overflow occurs. Upon successful address resolution, the system processes and transmits all queued packets in sequence. Additionally, the protocol implements a robust retry mechanism, featuring systematic retransmission of neighbor solicitation messages at specified intervals (with a recommended value of 1,000 ms) while enforcing strict rate-limiting measures. In cases where address resolution fails after a specified number of attempts (typically three), the protocol requires the system to notify upper layers by generating ICMP destination unreachable messages for all packets outgoing in the queue.

The NDP specification provides comprehensive implementation guidelines that effectively balance performance requirements with security considerations. However, it is crucial to note that these security guarantees are inherently limited to the data link layer. When examining the broader context of the entire TCP/IP network stack, as our subsequent analysis will reveal in detail, this apparent security can be significantly compromised due to subtle but dangerous vulnerabilities arising from complex cross-layer interactions.

4.2 The Implementation of Address Resolution in Linux and FreeBSD

Linux. In Linux systems, the neighbor subsystem module is responsible for processing address resolution. This module incorporates ARP and NDP at the lower layer and employs a unified state machine and cache table at the upper layer to control the operation of address resolution processes. The process encompasses various aspects, including neighbor entry states, neighbor cache management, and pending packet management. The state transition flow, which is closely linked to the SocketFilled attack, is illustrated in Figure 4.

Specifically, the process begins when packets are sent to a neighbor IP address without an existing neighbor entry. If the number of existing neighbor entries has not reached the maximum limit (Parameter 1), the neighbor subsystem creates and initializes a new neighbor entry. The outgoing packet is then queued in this neighbor entry, and the first neighbor resolution request is sent, transitioning the state to NUD_INCOMPLETE. In this state, the Linux neighbor subsystem sends resolution requests at a fixed rate (Parameter 2) until reaching the maximum retry attempts (Parameter 3). If a response is received before reaching the maximum attempts, neighbor resolution succeeds, all queued packets are transmitted, and the

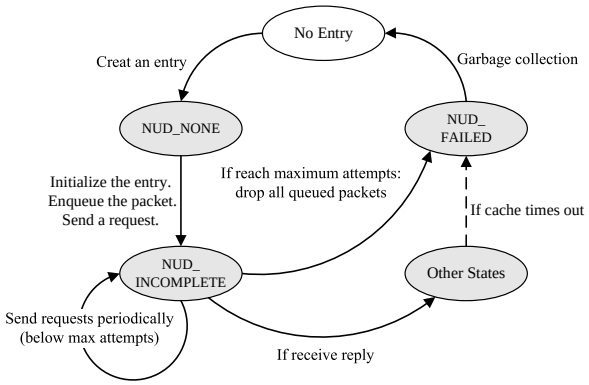


Figure 4: State Transition Diagram of the Linux Neighbor Subsystem

entry transitions to the subsequent cache timeout management state. If no response is received, the resolution fails, all packets in the neighbor entry's queue are discarded, and the neighbor entry state changes to NUD_FAILED, awaiting recycling when the number of neighbor entries approaches the upper limit. When packets are sent to an IP address with an existing neighbor entry in the NUD_INCOMPLETE state, they are directly added to the neighbor entry's pending packet queue. If the queue length exceeds the limit (Parameter 4), the oldest packet is replaced.

In general, Linux's implementation adheres to the NDP protocol specifications while not violating ARP specifications, despite the latter's relative lack of concrete guidelines. This implementation ensures data link layer security. However, Linux features a distinctive UDP implementation: UDP sockets maintain a send buffer size counter (Parameter 5), where unsent packets queued in neighbor entries continue to occupy the socket's send buffer allocation. Fundamentally, UDP was designed as a best-effort protocol that should not require send buffers for retransmission operations. Linux's design choice to maintain buffer accounting seems to align with the patterns used in TCP implementation. However, this architectural decision introduces vulnerabilities exploitable by the SocketFilled attack, which will be analyzed in detail in the following subsection.

FreeBSD. FreeBSD implements distinct logical structures for ARP and NDP protocols. In its ARP implementation, the initial packet transmission to an IP address without a corresponding neighbor entry triggers a resolution request with a 20-second response timeout period. Within this interval, each subsequent packet targeting the same neighbor generates an additional resolution query, subject to a rate-limiting mechanism that enforces the maximum query frequency. For packets exceeding the ARP retry count threshold, although they enter the pending packet queue, the system simultaneously returns a neighbor unreachable error to the upper-layer applications. Regarding pending packet management and NDP state transitions, FreeBSD employs an implementation similar to Linux's approach, conforming to NDP specifications.

With respect to UDP implementation, while FreeBSD also maintains the concept of UDP send buffers, their functionality is limited

to constraining UDP packet size maximums rather than accounting for unsent UDP packets. As our subsequent analysis will demonstrate, this architectural decision achieves TCP implementation consistency without introducing the cross-layer interaction vulnerabilities that can arise from more complex buffer management schemes.

4.3 Security Vulnerabilities in Linux's Integration of Address Resolution and UDP Pending Packet Management

Table 1 shows the critical kernel parameters in Linux associated with the SocketFilled attack. The UDP socket pending packet management mechanism illustrated in Section 4.2 requires that pending packets awaiting address resolution occupy space in the socket's send buffer. This enables the attack process described in Section 3.2: if pending packets fill the socket's send buffer, services relying on that socket for network communication become unresponsive until the neighbor entry's queue is cleared due to the timeout. This vulnerability requires that the total available space in the neighbor queue is no less than the total send buffer size of the socket used by the service, as shown in Formula (1).

$$\texttt{gc\_thresh3} \times \texttt{unres\_qlen\_bytes} \geq \texttt{wmem\_default} \quad (1)$$

Our analysis demonstrates that the default values of these parameters satisfy this condition, allowing a single unresolvable neighbor IP address to occupy a socket's send buffer space. In practice, applications can control the right-hand side of the inequality and the number of unresolvable neighbor entries depends on specific circumstances. This modifies the attack condition to Formula (2).

$$\texttt{\#\_of\_unresolvable\_neighbor} \times \texttt{unres\_qlen\_bytes} \geq \texttt{\#\_of\_sockets} \times \texttt{socket\_buffer\_size} \quad (2)$$

In Section 5, we will test various software applications with different configurations. The results demonstrate that almost all software is vulnerable to the SocketFilled attack, with the actual impact duration equal to the neighbor entry's queue timeout period, expressed as $\texttt{mcast\_solicit} \times \texttt{retrans\_time\_ms}$.

5 Experiments and Results

In this section, we evaluate the practical impact of the SocketFilled attack by examining its effectiveness across different systems and various UDP-based services. We assess the feasibility of SocketFilled on four mainstream operating systems and confirm that the latest version of Linux is affected. Furthermore, we conduct a comprehensive evaluation of nine mainstream implementations of two UDP protocols, including the impact of software parameters on attack efficacy, the theoretical and actual outcomes of the attacks, and the specific behavior of each software under attack. Overall, our experiments demonstrate that the SocketFilled attack poses a significant threat to UDP-based services on Linux.

Experiment Setup. To avoid impacting real-world services, we set up a simulated environment in our laboratory to conduct the attack tests. Specifically, we deployed test machines running target services in a controlled network, along with two additional hosts serving as the attacker and legitimate user. We used private IP addresses to simulate a LAN environment. Our environment did not deploy source address verification mechanisms to align with the capabilities required by the attacker. For IPv4, we used a /24 subnet, capable of providing up to 253 unresolvable neighbor IP addresses, with one IP allocated to each of the victim, attacker, user, and network gateway. For IPv6, we employed a /64 subnet, which can offer up to $2^{64} - 3$ unresolvable neighbor IP addresses. We then executed the attack procedure as described in Section 3.2 for each test target.

5.1 Vulnerable Operating System

We selected four mainstream desktop and server operating systems: Windows, Linux, macOS, and FreeBSD, and conducted black-box testing on their latest versions. The specific versions and test results are presented in Table 2. Specifically, we ran a basic UDP Echo service on each operating system with default system configurations, then performed the SocketFilled attack by sending extensive UDP packets to this UDP Echo service at a rate of 10,000 queries per second (qps). The attack used 253 unresolvable IP addresses in both IPv4 and IPv6 as spoofed source addresses. Throughout the attack, we monitored the status of the service and the utilization of the sending buffer.

The results indicate that among all four operating systems tested, Linux is vulnerable to this attack: the target UDP Echo service became almost immediately unresponsive after the attack began, with the sending buffer being completely filled, which aligns with our theoretical expectations. Moreover, this vulnerability can readily affect all versions since the introduction of the `unres_qlen_ bytes` parameter in Linux 3.3. As discussed in Section 4.3, this configuration makes the neighbor entry queue length default to the same size as the socket send buffer, allowing a single unresponsive neighbor to consume the entire socket send buffer. Prior to Linux 3.3, the neighbor entry queue length defaulted to three packets, requiring approximately 85 unresponsive neighbors to fill a socket send buffer, which increased the difficulty for attackers. FreeBSD is not affected due to its lack of a true send buffer. For Windows and macOS, being closed-source systems, we cannot analyze their underlying implementations.

5.2 Vulnerable Service Software

Service and Software List. To evaluate the potential impact of the SocketFilled attack on application-layer protocols, we selected two significant UDP-based application-layer protocols, DNS and QUIC, as our test subjects. The DNS protocol, which maps domain names to IP addresses, serves as fundamental Internet infrastructure and is widely deployed. The QUIC protocol, designed to address TCP's limitations, is a new transport-layer protocol used for HTTP/3 transmission and has gained support from mainstream browsers. In terms of software implementations, we examined seven DNS software applications serving as recursive resolvers or authoritative servers, along with two QUIC(HTTP/3) server implementations, as shown in Table 3.

Software Parameters and Theoretical Calculation. We installed these software applications on an Arch Linux host running Linux kernel 6.6. The host is equipped with a 20-core Intel CPU. We ran

Table 1: Parameters Affecting the SocketFilled Attack in the Linux Kernel.

No.	Parameter Name	Description from Linux Kernel Document or Man Page	Default Value
1	`gc_thresh3`	Maximum number of non-PERMANENT neighbor entries allowed	1,024
2	`retrans_time_ms`	The number of milliseconds to delay before retransmitting a request	1,000
3	`mcast_solicit`	The maximum number of attempts to resolve an address by multicast/broadcast before marking the entry asunreachable	3
4	`unres_qlen_bytes`	The maximum number of bytes which may be used by packets queued for each unresolved address by other network layers	212,992
5	`wmem_default`	The default setting (in bytes) of the socket send buffer	212,992

Table 2: SocketFilled Experiment Results of Operating Systems.

Operating System	Version	Vulnerable
Windows Server	2022	✗
Linux	6.6	✓
macOS	15.0	✗
FreeBSD	14.1	✗

these applications with their default configurations and observed the number of sockets they utilized. Additionally, we employed *strace* [44] to monitor system calls during software execution to verify whether the applications used custom socket send buffer sizes. After obtaining the necessary software parameters through this process, we calculated the practical requirements for the attack based on these parameters and the previously described attack principles.

Specifically, we calculated the total available send buffer size by multiplying the number of sockets by the buffer size per socket. Then, using the maximum possible response packet size that an attacker could trigger (1,232 bytes for DNS responses [11] and 1,322 bytes for QUIC handshake responses based on PMTU in our experiment environment), we computed the required number of response packets, which is equal to the number of request packets an attacker needs to send. We also calculated the required number of unresolvable neighbor IPs based on the space that each unresolvable neighbor can occupy (i.e., the neighbor queue size).

The software parameters and test results are presented in Table 3. Among all 10 measurements across 9 software applications (with BIND serving as both a DNS recursive resolver and authoritative server), BIND and Knot DNS utilized a number of sockets equal to the CPU core count, as they employ multi-threading by default, allocating one socket per thread through Linux port reuse. NSD and Caddy configured larger send buffer spaces using the `setsockopt` system call. Nevertheless, the required number of unresolvable IP addresses for all software remained within feasible ranges, indicating that all tested software implementations are potentially vulnerable to the SocketFilled attack.

5.3 Practical Attacks on Service Software

Experiment Setup. In addition to using the basic experimental environment, we implemented necessary configurations to ensure normal service operation and enhance attack efficiency. For the DNS recursive resolvers, we configured a large DNS record under our controlled domain name to trigger maximum-sized DNS responses. Similarly, for the DNS authoritative servers, we configured a comparably large record. For the QUIC servers, we implemented a self-signed certificate to enable normal operation.

To evaluate the actual attack effectiveness, we first performed normal requests to confirm the victim's response packet size. Then, we calculated the required number of attack packets based on the parameters determined in Section 5.2. The SocketFilled attack was then executed at `3.5-second` intervals three times, with the attack impact duration being the default value of 3 seconds. In other words, we maintained a 0.5-second window after each attack, during which the victim should theoretically be able to conduct normal packet transmission. We monitored the victim's actual socket send buffer utilization, along with the packet loss rate (with a 5-second wait time) and latency experienced by legitimate users in order to assess the impact of the attack.

Furthermore, we also tested the effectiveness of the attack under rate limiting defenses, which are usually deployed to counter a wide range of DDoS attacks. Specifically, we deployed rate-limiting policies on an intermediate node within the experimental environment to emulate network-level rate-limiting conditions. Following the node's rate-limiting enforcement, attack traffic reaching the victim server was throttled within discrete thresholds, including 10, 100, and 1000 qps.

Practical Attack Results. The attack results are presented in Table 5. We recorded the packet loss rates of the target software during the attack period and categorized the software behaviors into four patterns. Among all ten test cases, eight exhibited significant packet loss, with periods of complete unresponsiveness during the attack. Of the two software implementations that showed no packet loss, NSD successfully defended against the SocketFilled attack through its default prefix-based Response Rate Limiting (RRL) mechanism. However, when RRL was disabled, NSD also experienced packet loss and unresponsiveness. While Caddy did not exhibit packet loss, it demonstrated significant response latency during the attack. In summary, nearly all tested software implementations were affected by the SocketFilled attack, demonstrating a significant DoS impact.

Table 3: Parameters and Theoretical Calculation Results of Service Software.

Service Software				Parameter		Theoretical Calculate		
Service	Role	Software	Version	Socket (#)[1]	Socket Send Buffer (B)[2]	Total Send Buffer (B)	Attack Packet (#)	Unresolvable IP (#)
DNS	Recursor	BIND	9.20.2	20	212,992	4,259,840	3,458	20
		Unbound	1.21.1	1	212,992	212,992	173	1
		PowerDNS Recursor	5.1.2	2	212,992	425,984	345	2
		Knot Resolver	5.7.4	1	212,992	212,992	173	1
	Auth	BIND	9.20.2	20	212,992	4,259,840	3,458	20
		NSD	4.10.1	1	2,097,152	2,097,152	1,703	10
		PowerDNS	4.9.2	1	212,992	212,992	173	1
		Knot DNS	3.4.0	20	212,992	4,259,840	3,458	20
QUIC	Sever	Caddy	2.8.4	1	14,680,064	14,680,064	10,819	69
		Nginx	1.27.2	1	212,992	212,992	178	1

[1] 20 is the CPU core count, which means the software utilizes the same number of sockets as CPU cores.
[2] 212,992 indicates that the software utilizes the system default settings.

Table 5: Attack Experiment Results of Service Software.

Software	Avg. Packet Loss Rate		Software Behavior Type
	IPv4	IPv6	
BIND(Recursor)	41.1%	41.0%	Retry
Unbound	88.5%	88.4%	Poor Retry
PowerDNS Recursor	100.0%	100.0%	Unmoved
Knot Resolver	100.0%	100.0%	Unmoved
BIND(Auth)	53.8%	79.0%	Retry
NSD[1]	77.3%	78.0%	Poor Retry
PowerDNS	100.0%	100.0%	Unmoved
Knot DNS	100.0%	100.0%	Unmoved
Caddy[2]	0.0%	0.0%	Latency
Nginx	100.0%	100.0%	Unmoved

[1] We disabled NSD's Response Rate Limiting (RRL), which is enabled by default and can defend against SocketFilled attacks.
[2] Although the SocketFilled attack did not cause Caddy to drop packets directly, it resulted in noticeable service delays, which we define as *Latency*.

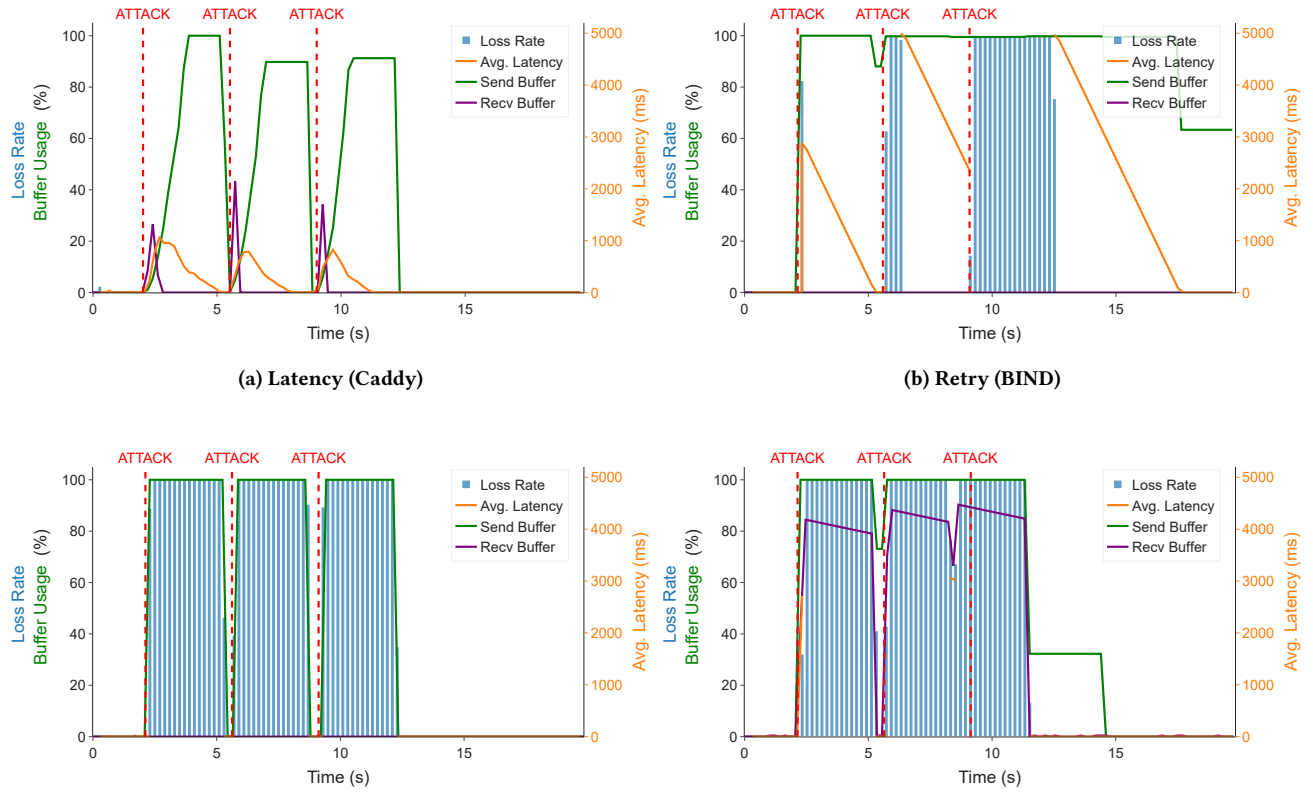
Software Behavior Patterns. Figure 5 illustrates the four behavioral patterns exhibited by software systems under the SocketFilled attack.

Latency: In this behavioral pattern, the socket send buffer utilization increases during the attack but does not reach full capacity, and rapidly decreases after the attack impact period ends. Instead of packet loss, this pattern exhibits response latency. This pattern was only observed in Caddy, likely due to its QUIC protocol implementation's ability to adjust send buffer utilization based on packet transmission status, preventing buffer exhaustion through mechanisms such as congestion control.

Retry: The send buffer becomes fully occupied after the first attack, but due to the software's retry mechanism, the majority of packets can be transmitted during the 0.5-second window following the impact of the attack, as evidenced by the observed latency variations (Figure 5(b)). In subsequent attacks, as packets destined for unresolvable neighbors are also retried, the software's retry capability gradually depletes. During the second attack, responses are only transmitted after two clearances of the neighbor entry queue, with some packets considered lost due to delays exceeding 5 seconds. During the third attack, responses are only transmitted after three clearances of the neighbor entry queue, resulting in all packets being considered lost. It should be noted that if we conduct a sustained attack without preserving window size, the victim will be unable to send packets.

Unmoved: This pattern aligns most closely with theoretical expectations: without any retry mechanisms, the software's send buffer is rapidly filled upon attack initiation, resulting in a 100% packet loss rate and complete service unresponsiveness. Once the attack impact period ends and the neighbor entry queue is cleared, the buffer immediately empties and service functionality returns to normal.

Poor Retry: In this pattern, the software implements retry mechanisms, but failed transmissions affect the rate at which requests are retrieved from the receive buffer. During the first attack, the behavior resembles *Unmoved*. However, after the second attack's impact period ends, retries of previously failed transmissions continue to occupy the send



(a) Latency (Caddy)

(b) Retry (BIND)

(c) Unmoved (Knot & PowerDNS)

(d) Poor Retry (Unbound & NSD)

Figure 5: Four Patterns of Software Behavior. We conduct attacks at 3.5-second intervals, with the attack effects persisting for 3 seconds, leaving a 0.5-second window to observe the software's behavioral patterns. We monitor the packet loss rate, average response latency, and the utilization of both send and receive buffers in the software's sockets. Responses with a latency exceeding 5 seconds are considered packet loss.

buffer, while normal requests subsequently fill the receive buffer. The third wave of attack packets cannot enter the socket's receive buffer and thus fail to trigger application responses. Effectively, the application is not impacted by the attacker's third attempt but rather becomes compromised by its own retry mechanism.

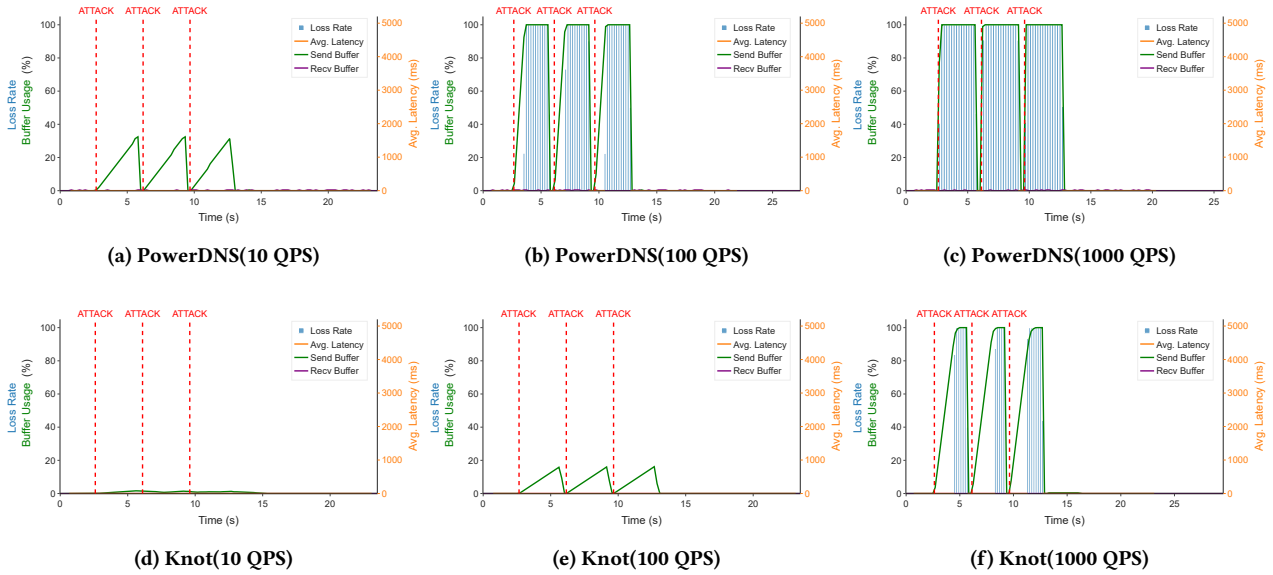
Rate Limiting Attack Results. Figure 6 shows the experimental results of PowerDNS and Knot, two representative DNS software, under varying rate-limiting configurations. The attack becomes effective when the attack rate can nearly fill the sending buffer within the attack window; otherwise, it only leads to an increase in socket buffer occupancy. For PowerDNS software, an overall rate limit of 100 QPS is sufficient to enable the attack, which impacts most mainstream public DNS services, as their rate limits generally remain above 100 qps [25].

6 Estimation of Vulnerable Service Population

In this section, we first demonstrate how certain protective measures deployed by cloud providers have inadvertently provided defense against the SocketFilled attack. Therefore, we focus on evaluating the impact scope of SocketFilled attacks on services not hosted on cloud platforms. Moreover, we estimate the vulnerability status of open resolvers, authoritative servers and QUIC (HTTP/3) servers from the Tranco Top100K based on the number of unresolvable neighbors. The results show that a significant number of Internet services not deployed on cloud servers are vulnerable to SocketFilled attacks.

6.1 Cloud Providers' Inadvertent Defenses Methods

We observed that cloud providers (CPs) typically organize their hosted VPS using approaches different from traditional LANs, such as software-defined networking, to enhance scalability, resulting in different address resolution mechanisms [32, 40]. We examined how popular cloud service providers handle address resolution in both IPv4 and IPv6 network stacks and summarized their methods for handling address resolution requirements in Table 7.



(a) PowerDNS(10 QPS) (b) PowerDNS(100 QPS) (c) PowerDNS(1000 QPS)

(d) Knot(10 QPS) (e) Knot(100 QPS) (f) Knot(1000 QPS)

Figure 6: Experimental results under rate limiting.

Based on web searches and cloud provider market share [34], we selected six representative cloud service providers and manually registered trial accounts. We created Linux VPS instances on dual-stack networks supporting both IPv4 and IPv6. Then, we examined the network configurations and routing settings of these VPS instances and conducted ARP and NDP test scans on IPv4 and IPv6 networks using *arpscan* and *ndisc6* tools, respectively.

Table 7: Cloud Provider's Methods for Handling Address Resolution.

Cloud Provider	Handling Method	
	IPv4	IPv6
AWS	Invisible Public IP	/128 Subnet
Azure	Invisible Public IP	Invisible Public IP
Google	Invisible Public IP	/128 Subnet
Digital Ocean	ARP Spoofing	NDP Spoofing
Linode	ARP Spoofing	Single-route table
Alibaba	Invisible Public IP	NDP Spoofing

Finally, we found that all cloud providers employ similar approaches to handle address resolution with a common objective: preventing VPS instances from directly sending packets to neighbors and enforcing packet forwarding through routers. Multiple methods are implemented to achieve this goal:

Invisible Public IP: Cloud providers often assign only private addresses to VPS instances and map public addresses to private ones through mechanisms like address translation. In this scenario, attackers cannot identify their victims' private addresses, and even if they manage to guess these addresses, forged request packets cannot be routed to the victims over the Internet.

ARP/NDP Spoofing: Cloud service providers may intercept address resolution requests from VPS instances within the local network by responding with the router's link-layer address for any neighbor query. This prevents SocketFilled attacks from generating packets that need to wait for address resolution responses, thus preventing the occupation of send buffer space that could impact service operations.

/128 Subnet and Single-route Table: The absence of directly connected routes in the routing table, implemented through /128 subnets and single-route tables, eliminates the need for address resolution. VPS instances forward packets directly to the router instead, effectively preventing SocketFilled attacks from being executed.

Although these measures deployed by cloud service providers were not specifically designed to defend against the SocketFilled attack, they effectively prevented attackers from exploiting unresolvable addresses to fill up the UDP buffer, thereby mitigating the attack's impact. This provides valuable insights for developing our mitigation strategies.

6.2 Estimation of Vulnerable Service in the Wild

Service List. As mentioned in Section 5, we evaluated three types of UDP services on the Internet: DNS recursive resolvers, DNS authoritative servers, and QUIC servers. For DNS recursive resolvers, we scanned UDP port 53 across the IPv4 address space using ZMap [13] and our domain names to obtain an updated list of open DNS resolvers. For DNS authoritative services and QUIC servers, we selected DNS authoritative servers and HTTP/3-enabled web servers supporting QUIC from the Tranco Top 100K websites.

Given that cloud providers implement various measures that can effectively defend against SocketFilled attacks, we utilized ASdb [48], a research database that provides industry classifications for autonomous systems, to exclude target services deployed on cloud providers.

Methodology. As analyzed in Section 4, the feasibility of the attack depends on the number of sockets used, Linux system parameters, and the number of exploitable unresolvable neighbor IPs. While the first two factors are difficult to evaluate through network measurements, the insecurity of most software and default Linux configurations has been demonstrated in the experiments in Section 4 and 5.

Moreover, we assess the scale of affected hosts in the wild by detecting the number of unresolvable neighboring IP addresses of target hosts. The liveness verification of arbitrary hosts remains an open research challenge, and thus our method employs the following workflow to preserve the objectivity of evaluation results.

For IPv4 networks, we employed a multi-step approach:

(1) We first identified the subnet in which the target host resides. Specifically, we used `traceroute` [45] to find the subnet containing target hosts sharing the last-hop router. This estimation is conservative when multi-path load balancing exists.
(2) We then performed `traceroute` again to every host within the /24 subnet of each target host IP to identify the last-hop IP of live hosts (those responding to traceroute requests). We calculated the largest subnet containing the target host that excludes hosts with different last-hop IPs from the target host.
(3) We tested the number of hosts within this subnet that do not respond to ICMP ping requests to assess whether the target host is affected.
(4) We used the number of non-responsive hosts within this subnet as the count of unresolvable neighbor IPs. When the count exceeds the threshold `T`, the target host is deemed affected. Our experimental results show that if the number of non-responsive hosts within the same subnet exceeds 2, the target host will be affected by the attack. Thus, we established 2 as the threshold value.

For IPv6 networks, since protocol standards [41] suggest that the most specific prefix assigned to endpoints is /64, each endpoint potentially has access to a considerable number of unresolvable neighbor addresses that could be exploited. In this case, random addresses can be used for attacks, confirming that attack requirements are met.

Our approach represents a conservative estimation based on two key assumptions: 1) Directly connected neighboring hosts share the same upstream router, and 2) Hosts that do not respond to ICMP Echo requests also do not respond to address resolution requests. The first assumption generally holds true in typically configured LANs, and given that we have filtered out targets from cloud service providers, this assumption can be considered reliable.

Note that network administrators may deploy ICMP filtering mechanisms, which could lead to inaccurate evaluation. However, such mechanisms are typically deployed at the gateway, so when they are in place, all hosts within the subnet will not respond to ICMP. We excluded these results from our evaluation to avoid overestimating the scope of the impact.

Considering these factors together, while there indeed exists a certain number of active hosts that do not respond to ICMP requests, which may lead to an underestimation of the host count, this is partially offset by our use of the largest possible subnet for estimation. Therefore, the overall estimation can be deemed credible and conservative.

Table 8: Statistical Data of Measurement Results.

Target	# of IP	# of IP not on Cloud	# of /24 subnet
Open Resolver	505,930	462,125	175,518 [1]
DNS Auth Server	1,384	1,306	579
QUIC Server	29,416	1,655	984

[1] We randomly selected 1,000 subnets for subsequent measurements.

Measurement Results. Table 8 presents the statistical information of the measurement targets. Due to the large volume of open resolvers, we randomly sampled one thousand /24 subnets for subsequent measurements to estimate the number of unresolvable neighbors through traceroute.

Figure 7 illustrates the measurement results. Based on the experimental results in Section 5, we consider target services with no fewer than two unresolvable neighbors to be vulnerable. According to our measurements, among services not deployed on cloud platforms, **17.3%** of open resolvers, **54.3%** of DNS authoritative servers, and **3.8%** of QUIC servers are potentially susceptible to SocketFilled attacks. Some well-known websites have a significant number of non-responsive neighboring IPs, such as Bing (30), Amazon (195), and Shopee (210), making them potential targets for SocketFilled attacks. Overall, there is a considerable number of potential targets for the SocketFilled attack in the wild.

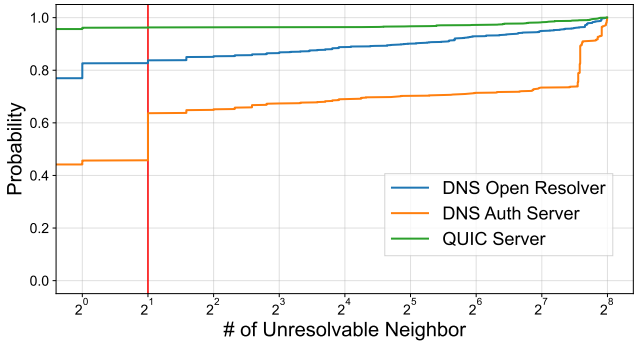


Figure 7: The Cumulative Distribution of Measurement Results of Unresolvable Neighbor Numbers.

7 Discussion

In this section, we first analyze the impact of the SocketFilled attack on TCP connections and routing forwarding in Linux. Subsequently, we propose several mitigation strategies against the SocketFilled attack and report our responsible disclosure. Finally, we elaborate on the ethical considerations throughout our research process.

7.1 SocketFilled Attacks on TCP and Routing Forward

Despite sharing the same network stack and neighbor subsystem, TCP and routing forwarding in the Linux kernel network stack employ different packet transmission mechanisms compared to UDP, making these two functionalities immune to the SocketFilled attack in Linux systems.

TCP. Due to its connection-oriented and reliability-ensuring characteristics, TCP requires a three-way handshake for connection establishment. The SocketFilled attack, which relies on IP spoofing, cannot receive server responses to complete the three-way handshake and establish the connection, thus limiting its impact to only the SYN queue. In this scenario, the SocketFilled attack becomes functionally similar to a SYN flood attack. Therefore, the SocketFilled attack can be effectively mitigated by the SYN Cookie mechanism, which is enabled by default in Linux systems.

Routing Forward. Unlike transport layer protocols such as UDP that serve upper-layer applications, Linux packet forwarding performs routing lookups directly at the network layer without consuming any socket send buffer space. Due to the inherent limitations of the Linux neighbor subsystem, including neighbor entry quantity restrictions, neighbor entry garbage collection, and the neighbor entry queue's First-In-First-Out (FIFO) replacement mechanism, the SocketFilled attack cannot impact the routing forwarding functionality in the absence of shared buffer space resources.

7.2 Impact of Source Address Validation

Source Address Validation (SAV), also referred to as ingress filtering, denotes the technical methodology by which spoofed packets are discarded at network boundaries (i.e., routers) based on routing configurations. Within our contextual framework, ingress filtering does not differentiate between inbound and outbound traffic flows, as from the router's perspective, the traffic subject to filtration is invariably ingress traffic. The terminological distinction between inbound/outbound SAV refers specifically to ingress filtering executed at the destination/source network boundaries, respectively.

Correctly implemented inbound SAV, capable of filtering packets originating from internal addresses, has been empirically demonstrated to effectively mitigate the attacks we propose. A prior research [37] has conducted systematic measurements regarding inbound SAV deployment and substantiated the absence of implementation across a significant proportion of Autonomous Systems (ASs). This empirical finding aligns with CAIDA's longitudinal comprehensive monitoring (encompassing both inbound and outbound) [5]. Concerning the reasons for the lack of inbound SAV deployment, the prior research [37] also conducted several hypotheses and investigations, including operator resource constraints, AS types, and practical authority issues.

In summary, while proper deployment of Source Address Validation can provide defense against the SocketFilled attack, there are significant deficiencies in its current deployment due to various reasons, which result in the SocketFilled attack having a widespread impact.

7.3 Mitigation Solutions

Based on our testing and analysis, we propose corresponding mitigation measures for network administrators, Linux kernel maintainers, and software developers to address the SocketFilled attack.

Source Address Validation. For public network service operators, implementing source address validation measures, such as Reverse Path Filtering, at edge routers serves as an effective defense against various source address spoofing-based attacks, including the SocketFilled attack. Furthermore, promoting the deployment of source address verification at the Autonomous System level represents a crucial initiative in mitigating such attacks.

Direct Traffic Transmission to Routers. As demonstrated in Section 7.1 , the SocketFilled attack is ineffective against routing forwarding functionality. Therefore, directing traffic straight to routers through ARP/NDP spoofing or specialized routing configurations, similar to the measures deployed by cloud service providers in Section 6.1, enables effective defense against SocketFilled attacks through router-based forwarding and address resolution. Although such configurations may generate unnecessary traffic for routers and switches, these network components usually have sufficient performance redundancy to handle the overhead.

Additionally, performance-critical software may employ kernel bypass solutions like DPDK, which typically lack the concept of sockets and require manual handling of packet transmission buffers and address resolution requests, thus providing potential opportunities for defending against SocketFilled attacks.

Modifying Linux Kernel Parameters. Although the Linux kernel adopts potentially risky parameter settings for compatibility and performance considerations, these parameters can be modified by users. Users can adjust kernel parameters based on their specific deployment scenarios — for example, adjusting the maximum number of neighbor entries (`gc_thresh3`) according to the actual subnet size and host count, and modifying the socket send buffer size (`wmem_default`) based on available memory capacity. By adjusting these parameters to violate Formula 2, the SocketFilled attack can be completely prevented. Moreover, only increasing the socket send buffer size can elevate the difficulty level for potential attackers.

Adjusting Linux Kernel Protocol Stack Mechanisms. Modifying the kernel network stack mechanisms to prevent unsafe occupation of socket sending buffer space by the neighbor subsystem represents the most fundamental solution to this security problem despite it potentially leading to extensive code modifications or introducing new issues. FreeBSD's implementation could serve as a reference. However, according to discussions with Linux security officers, the Linux kernel must choose among various potentially problematic implementations. While FreeBSD's implementation is immune to SocketFilled attacks, it may introduce other vulnerabilities. Furthermore, Linux's complex code dependencies make it

particularly challenging to modify such low-level system mechanisms.

Improved Software Implementation. Software implementations can help mitigate SocketFilled attacks even if the vulnerability exists in the Linux kernel. As demonstrated by the experimental results in Section 4.1, while complete prevention may not be possible, software can mitigate the impact of SocketFilled attacks through various measures: implementing robust retry mechanisms, enabling prefix-based rate limiting, utilizing multiple sockets, requesting larger socket send buffers, and actively managing buffer space. Furthermore, for protocols that support both TCP and UDP operations, such as DNS, enabling TCP services alongside UDP can indirectly mitigate the attack's impact.

7.4 Disclosure and Research Significance

The research motivation originates from large-scale real-world attack activities observed within our collaborating ISP network. Our systematic analysis exposes the effectiveness of cross-layer mechanisms in DoS attacks in Linux. While similar user-reported issues in certain software [29] were identified in Linux mailing lists, they received inadequate community attention. Our extensive experimental results demonstrate the attack's broad impact on UDP-based protocols in Linux. We re-engaged proactively with the Linux community; however, Linux kernel security officials maintained that mitigation should be achieved through edge-router source address validation. We also disclosed this vulnerability to affected upper-layer service software vendors. They consider this an issue with the Linux system, and there is relatively little that software can do about it.

Although this vulnerability is not a new issue within the Linux kernel community, this research is the first to identify its security implications and propose corresponding threat models and exploitation methods. Our software testing confirms that almost all UDP-based service applications and a significant number of active services (Section 6) are vulnerable to SocketFilled attacks. A recent study [10] and CAIDA's long-term measurements indicate that a significant number of IP address blocks remain vulnerable to IP spoofing even though address verification measures deployed at border gateways are indeed effective countermeasures against all source address spoofing-based attacks. Furthermore, in edge LANs such as corporate networks and airport networks, the SocketFilled attack can also pose threats to UDP-based services like DNS and NTP within the local network.

7.5 Ethical Considerations

Since our research involves serious security vulnerabilities and active Internet services, we conducted comprehensive ethical considerations to prevent any impact on real-world networks. We adhered to the Menlo Report's ethical principles [20] and the best network measurement practices [38].

For local testing (Section 5), we installed all test software on hosts under our control and conducted experiments within our controlled local network environment. The number of DNS infrastructure requests during testing was strictly regulated, and all domain names and IP addresses involved were owned by us.

For Internet-scale measurements (Section 6), including scanning of open resolvers, probing authoritative servers of the Tranco Top 100K websites, and their QUIC web servers, we strictly limited our probing rate to 5,000 packets per second to prevent any overhead on target networks and hosts. For traceroute measurements, we employed randomized scanning patterns to avoid impacting specific paths. We also configured PTR records and websites to demonstrate our measurement purposes, and received no complaints throughout the measurement process.

8 Conclusion

In this paper, we investigate systematically the cross-layer DoS attack that exploits vulnerabilities in the interactions across layers within the Linux kernel network stack. By analyzing address resolution specifications and their implementations in Linux and FreeBSD, we identify insecure interactions between the link layer and the transport layer in the Linux kernel network stack that enable the SocketFilled attack. Our experiments in controlled environments demonstrate that the SocketFilled attack poses a general threat to UDP services running on Linux, including DNS and QUIC. Internet-scale measurements reveal that a significant number of Internet services, including well-known websites like Bing and Amazon, are potentially vulnerable to the SocketFilled attack.

Acknowledgments

We thank all anonymous reviewers for their valuable and constructive feedback. This work is supported by the National Key Research and Development Program of China (No. 2023YFB3105600) and the National Natural Science Foundation of China (62102218). Baojun Liu is the corresponding author.

References

[1] Akamai. 2024. State of the Internet Reports. https://www.akamai.com/security-research/the-state-of-the-internet.
[2] Bill Toulas. 2023. New 'HinataBot' Botnet Could Launch Massive 3.3 Tbps DDoS Attacks. https://www.bleepingcomputer.com/news/security/new-hinatabot-botnet-could-launch-massive-33-tbps-ddos-attacks/.
[3] Bill Toulas. 2023. New 'HTTP/2 Rapid Reset' Zero-Day Attack Breaks DDoS Records. https://www.bleepingcomputer.com/news/security/new-http-2-rapid-reset-zero-day-attack-breaks-ddos-records/.
[4] Robert T. Braden. 1989. Requirements for Internet Hosts - Communication Layers. RFC 1122. doi:10.17487/RFC1122
[5] CAIDA. 2025. State of IP Spoofing. https://spoofer.caida.org/summary.php.
[6] Weiteng Chen and Zhiyun Qian. 2018. {Off-Path} {TCP} Exploit: How Wireless Routers Can Jeopardize Your Secrets. In *27th USENIX Security Symposium (USENIX Security 18)*. 1581–1598.
[7] Cloudflare. 2024. DDoS threat report for 2024 Q2. https://blog.cloudflare.com/ddos-threat-report-for-2024-q2/.
[8] Cloudflare. 2025. What Is a Distributed Denial-of-Service (DDoS) Attack? https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/.
[9] Amir Dahan. 2021. Business as Usual for Azure Customers despite 2.4 Tbps DDoS Attack. https://azure.microsoft.com/en-us/blog/business-as-usual-for-azure-customers-despite-24-tbps-ddos-attack/.
[10] Tianxiang Dai and Haya Shulman. 2021. SMap: Internet-wide Scanning for Spoofing. In *Proceedings of the 37th Annual Computer Security Applications Conference (ACSAC '21)*. Association for Computing Machinery, New York, NY, USA, 1039–1050. doi:10.1145/3485832.3485917
[11] DNS-ORAC. 2020. DNS flag day 2020. https://dns-violations.github.io/dnsflagday/2020/.
[12] Huayi Duan, Marco Bearzi, Jodok Vieli, David Basin, Adrian Perrig, Si Liu, and Bernhard Tellenbach. 2024. {CAMP}: Compositional Amplification Attacks against {DNS}. In *33rd USENIX Security Symposium (USENIX Security 24)*. 5769–5786.
[13] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. 2013. ZMap: Fast Internet-wide scanning and its security applications. In *22nd USENIX Security Symposium*.

[14] Dyn. 2016. DDoS Attack Against Dyn Managed DNS. https://www.dynstatus.com/incidents/nlr4yrr162t8.

[15] Emil Kiner and Tim April. 2023. Google Cloud Mitigated Largest DDoS Attack, Peaking above 398 Million Rps. https://cloud.google.com/blog/products/identity-security/google-cloud-mitigated-largest-ddos-attack-peaking-above-398-million-rps.

[16] Xuewei Feng, Chuanpu Fu, Qi Li, Kun Sun, and Ke Xu. 2020. Off-Path TCP Exploits of the Mixed IPID Assignment. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*. Association for Computing Machinery, New York, NY, USA, 1323–1335. doi:10.1145/3372297.3417884

[17] Xuewei Feng, Qi Li, Kun Sun, Yuxiang Yang, and Ke Xu. 2023. Man-in-the-Middle Attacks without Rogue AP: When WPAs Meet ICMP Redirects. In *2023 IEEE Symposium on Security and Privacy (SP)*. 3162–3177. doi:10.1109/SP46215.2023.10179441

[18] Mukesh Gupta and Alex Conta. 2006. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 4443. doi:10.17487/RFC4443

[19] Elias Heftrig, Haya Schulmann, Niklas Vogel, and Michael Waidner. 2024. The Harder You Try, The Harder You Fail: The KeyTrap Denial-of-Service Algorithmic Complexity Attacks on DNSSEC. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security (CCS '24)*. Association for Computing Machinery, New York, NY, USA, 497–510. doi:10.1145/3658644.3670389

[20] Erin Kenneally and David Dittrich. 2012. The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research. social science research network:2445102 doi:10.2139/ssrn.2445102

[21] Amit Klein. 2021. Cross Layer Attacks and How to Use Them (for DNS Cache Poisoning, Device Tracking and More). In *2021 IEEE Symposium on Security and Privacy (SP)*. 1179–1196. doi:10.1109/SP40001.2021.00054

[22] Qrator Labs. 2023. Q2 2023 DDoS Attacks Statistics and Overview.

[23] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society, San Diego, CA. doi:10.14722/ndss.2019.23386

[24] Weizhong Li, Kaiwen Shen, Run Guo, Baojun Liu, Jia Zhang, Haixin Duan, Shuang Hao, Xiarun Chen, and Yao Wang. 2020. CDN Backfired: Amplification Attacks Based on HTTP Range Requests. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, Valencia, Spain, 14–25. doi:10.1109/DSN48063.2020.00022

[25] Xiang Li, Dashuai Wu, Haixin Duan, and Qi Li. 2024. DNSBomb: A New Practical-and-Powerful Pulsing DoS Attack Exploiting DNS Queries-and-Responses. In *2024 IEEE Symposium on Security and Privacy (SP)*. 4478–4496. doi:10.1109/SP54263.2024.00264

[26] Xiang Li, Wei Xu, Baojun Liu, Mingming Zhang, Zhou Li, Jia Zhang, Deliang Chang, Xiaofeng Zheng, Chuhan Wang, Jianjun Chen, Haixin Duan, and Qi Li. 2024. TUDOOR Attack: Systematically Exploring and Exploiting Logic Vulnerabilities in DNS Response Pre-processing with Malformed Packets. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 185–185. doi:10.1109/SP54263.2024.00172

[27] Yuanjie Li, Hewu Li, Zhizheng Lv, Xingkun Yao, Qianru Li, and Jianping Wu. 2021. Deterrence of Intelligent DDoS via Multi-Hop Traffic Divergence. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21)*. Association for Computing Machinery, New York, NY, USA, 923–939. doi:10.1145/3460120.3484737

[28] Ziyu Lin, Zhiwei Lin, Ximeng Liu, Jianjun Chen, Run Guo, Cheng Chen, and Shaodong Xiao. 2024. {CDN} Cannon: Exploiting {CDN} {Back-to-Origin} Strategies for Amplification Attacks. In *33rd USENIX Security Symposium (USENIX Security 24)*. 5717–5734.

[29] Linux kernel netdev mailing list. 2019. Kernel UDP Behavior with Missing Destinations. https://www.spinics.net/lists/netdev/msg570519.html.

[30] Keyu Man, Xin'an Zhou, and Zhiyun Qian. 2021. DNS Cache Poisoning Attack: Resurrections with Side Channels. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21)*. Association for Computing Machinery, New York, NY, USA, 3400–3414. doi:10.1145/3460120.3486219

[31] Mousa Taghizadeh Manavi. 2018. Defense Mechanisms against Distributed Denial of Service Attacks : A Survey. *Computers & Electrical Engineering* 72 (Nov. 2018), 26–38. doi:10.1016/j.compeleceng.2018.09.001

[32] Microsoft Q&A. 2024. Is It Possible to Enable ARP Spoofing in Azure Virtual Networks? https://learn.microsoft.com/en-us/answers/questions/1522014/is-it-possible-to-enable-arp-spoofing-in-azure-vir.

[33] Michael Mirkin, Yan Ji, Jonathan Pang, Ariah Klages-Mundt, Ittay Eyal, and Ari Juels. 2020. BDoS: Blockchain Denial-of-Service. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*. Association for Computing Machinery, New York, NY, USA, 601–619. doi:10.1145/3372297.3417247

[34] Subhendu Nayak. 2024. The 10 Leading Cloud Service Providers of 2024. https://www.cloudoptimo.com/blog/the-10-leading-cloud-service-providers-of-2024/.

[35] Netscout. 2025. What Is a Slow Post DDoS Attack? https://www.netscout.com/what-is-ddos/slow-post-attacks.

[36] Hoai Viet Nguyen, Luigi Lo Iacono, and Hannes Federrath. 2019. Your Cache Has Fallen: Cache-Poisoned Denial-of-Service Attack. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*. Association for Computing Machinery, New York, NY, USA, 1915–1936. doi:10.1145/3319535.3354215

[37] Yevheniya Nosyk, Maciej Korczyński, Qasim Lone, Marcin Skwarek, Baptiste Jonglez, and Andrzej Duda. 2023. The Closed Resolver Project: Measuring the Deployment of Inbound Source Address Validation. *IEEE/ACM Trans. Netw.* 31, 6 (March 2023), 2589–2603. doi:10.1109/TNET.2023.3257413

[38] Craig Partridge and Mark Allman. 2016. Ethical Considerations in Network Measurement Papers. *Commun. ACM* 59, 10 (Sept. 2016), 58–64. doi:10.1145/2896816

[39] D. Plummer. 1982. An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826. doi:10.17487/RFC0826

[40] Teri Radichel. 2024. Why One of Your Favorite Pen Testing Techniques Doesn't Work on AWS.

[41] Rosalea Roberts, Geoff Huston, and Dr. Thomas Narten. 2011. IPv6 Address Assignment to End Sites. RFC 6177. doi:10.17487/RFC6177

[42] William A. Simpson, Dr. Thomas Narten, Erik Nordmark, and Hesham Soliman. 2007. Neighbor Discovery for IP version 6 (IPv6). RFC 4861. doi:10.17487/RFC4861

[43] Jared M Smith and Max Schuchard. 2018. Routing Around Congestion: Defeating DDoS Attacks and Adverse Network Conditions via Reactive BGP Routing. In *2018 IEEE Symposium on Security and Privacy (SP)*. 599–617. doi:10.1109/SP.2018.00032

[44] Strace. 2025. Strace. https://strace.io/.

[45] Traceroute. 2025. Traceroute for Linux. https://traceroute.sourceforge.net/.

[46] Junjie Xiong, Mingkui Wei, Zhuo Lu, and Yao Liu. 2021. Warmonger: Inflicting Denial-of-Service via Serverless Functions in the Cloud. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21)*. Association for Computing Machinery, New York, NY, USA, 955–969. doi:10.1145/3460120.3485372

[47] Wei Xu, Xiang Li, Chaoyi Lu, Baojun Liu, Haixin Duan, Jia Zhang, Jianjun Chen, and Tao Wan. 2023. TsuKing: Coordinating DNS Resolvers and Queries into Potent DoS Amplifiers. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*. Association for Computing Machinery, New York, NY, USA, 311–325. doi:10.1145/3576915.3616668

[48] Maya Ziv, Liz Izhikevich, Kimberly Ruth, Katherine Izhikevich, and Zakir Durumeric. 2021. ASdb: A System for Classifying Owners of Autonomous Systems. In *Proceedings of the 21st ACM Internet Measurement Conference (IMC '21)*. Association for Computing Machinery, New York, NY, USA, 703–719. doi:10.1145/3487552.3487853