

Analyzing Logic Vulnerabilities in DNS Response Pre-processing: From Kaminsky to TuDoor

Autret Lucas and Terrien Maxime

Index Terms—DNS Security, Logic Vulnerabilities, TuDoor Attack, Kaminsky Attack, SAD DNS, DNS Response Pre-processing

1 INTRODUCTION

The Domain Name System (DNS) is one of the most critical infrastructure of the modern Internet because of its function. Designed in the 1980s, this protocol translates human-readable domain names into IP addresses, making web navigation easier for users. However, its age and widespread adoption have made it a prime target for attackers seeking to compromise Internet communications.

Over the past two decades, DNS has been the subject of numerous cache poisoning attacks. The Kaminsky attack in 2008 revealed fundamental weaknesses in the protocol, leading to multiple patches including source port randomization. Despite these countermeasures, SAD DNS in 2020 demonstrated that side-channel vulnerabilities in operating systems could bypass existing protections. More recently, the TuDoor attack (2024) has unveiled a new attack surface: logic vulnerabilities in DNS response pre-processing, where inconsistent handling of malformed packets across implementations creates exploitable conditions.

This paper analyzes the evolution of DNS attacks and examines in detail the TuDoor attack methodology notably on cache poisoning. We will first present the DNS architecture in section 2 to get a better understanding of how it works. Then we will get an overview of the history of DNS attacks with Kaminsky and SAD DNS. After that, we will describe the TuDoor attack in section 3, its technical mechanisms, and comparative analysis with prior work. Finally, we will discuss the impact on DNS security and propose mitigation strategies in section 4.

2 STATE OF THE ART AND HISTORICAL OVERVIEW

2.1 How DNS Works

The Domain Name System (DNS) serves as a crucial component of the Internet infrastructure, that translates human-readable domain names into machine-readable IP addresses. As illustrated in Figure 1, the resolution process relies on a chain of interactions between several distinct components to locate the correct resource.

The resolution process begins when a client application, such as a web browser, needs to resolve a hostname. It uses the operating system's *stub resolver* to initiate a request. To optimize performance and reduce the latency, this request is first sent to a pre-configured **DNS Forwarder**, often integrated into local network devices like home Wi-Fi routers. If the forwarder does not have the answer in its cache, it forwards the query to a recursive resolver for further processing.

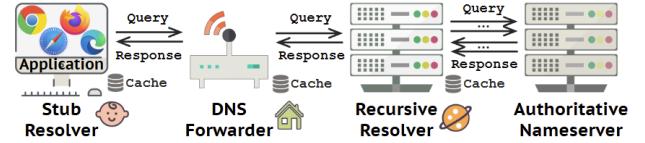


Fig. 1: General DNS resolver roles and domain name resolution process.

The **Recursive Resolver** plays an important role in the DNS resolution process. Upon receiving a query from the forwarder, it first checks its cache for a valid response. If the answer is not cached, the recursive resolver embarks on a systematic process to resolve the domain name. It begins by querying the **Root DNS Servers**, which provide referrals to the appropriate **Top-Level Domain (TLD) Servers** based on the domain's extension (e.g., .com, .org). The recursive resolver then queries the TLD servers, which in turn refer it to the **Authoritative DNS Servers** responsible for the specific domain. Finally, the authoritative server provides the requested IP address, which is relayed back through the chain to the original client.

2.2 Kaminsky attack (2008)

In 2008, security researcher Dan Kaminsky unveiled a critical vulnerability in the DNS protocol. The core issue was not just the lack of entropy in the *TxID*, but a technique that allowed an attacker to bypass the Time-To-Live (TTL) mechanism that was supposed to slow down cache poisoning attempts.

Prior to this discovery, if an attacker failed to poison a DNS cache, they would have to wait for the TTL of the cached record to expire before trying again. Kaminsky's attack exploited the fact that DNS resolvers would accept random queries for non-existent subdomains of a target domain (e.g., random123.target.com, random456.target.com). Since these subdomains do not exist in the cache, the recursive resolver is forced to query the authoritative nameserver, giving the attacker a infinite number of opportunities to flood the resolver with spoofed responses.

At the time, resolvers typically used a static source port for outgoing DNS queries, which meant that the only field an attacker needed to guess was the 16-bit *TxID*. This provided a search space of only 2^{16} (65,536) possible values. By sending a large number of spoofed DNS responses with different *TxID* values, the attacker

could eventually guess the correct one and successfully poison the cache in a matter of minutes.

Ultimately, the malicious response would be accepted by the resolver, which would then cache the incorrect mapping. This allowed the attacker to redirect users to malicious sites, intercept sensitive information, or launch further attacks. This discovery led to the implementation of **Source Port Randomization (SPR)** which increased the entropy to roughly 32 bits ($TxID$ + random 16-bit source port), making brute-forcing attacks significantly more difficult.

2.3 SAD DNS (2020)

The **SAD DNS (Side-channel AttackeD DNS) attack**, disclosed in 2020 by researchers from Tsinghua University and the University of California, Riverside, marked a critical regression in DNS security. It demonstrated a method to effectively resurrect the classic DNS cache poisoning attack by bypassing the primary mitigation implemented after the 2008 Kaminsky attack: **Source Port Randomization (SPR)**.

The success of SAD DNS relies on exploiting a subtle, yet pervasive, vulnerability in the networking stacks of modern operating systems: the predictable rate limit applied to outgoing **Internet Control Message Protocol (ICMP)** error messages, specifically the "Port Unreachable" message. This ICMP rate limit serves as a timing side-channel that allows an off-path attacker to significantly reduce the entropy of a DNS query.

Prior to this attack, SPR had increased query entropy from 16 bits (Transaction ID, $TxID$) to 32 bits ($TxID$ plus the random 16-bit source port). The attack uses the following sequence to infer the source port:

- 1) **Probe Emission:** The attacker sends a large burst of spoofed UDP probe packets targeting the victim DNS recursive resolver's port range. The source IP address of these probes is spoofed to that of the target authoritative name server.
- 2) **ICMP Trigger:** The resolver's kernel generates an ICMP "Port Unreachable" error message whenever a probe hits a closed port. Conversely, if the probe hits the active, open port currently used for the pending DNS query, the ICMP error is suppressed.
- 3) **Rate Limit Inference:** The key exploitation mechanism is the fact that the operating system applies a global rate limit to all outgoing ICMP errors. The attacker sends a final, "unspoofed" probe to a known closed port on the resolver, observing the response time.
 - If the preceding burst of spoofed probes hit enough closed ports to deplete the global ICMP quota, the final legitimate probe will experience response delay or suppression.
 - If the burst included a hit on the active DNS source port, the corresponding ICMP error was suppressed, leaving the global quota available.
- 4) **Source Port Derandomization:** By analyzing the timing and successful delivery of the final probe, the attacker can systematically infer which ports in the range are currently active. This process effectively derandomizes the 16-bit source port.
- 5) **Cache Poisoning:** With the source port identified, the remaining entropy is reduced to the 16-bit $TxID$, enabling

the attacker to easily brute-force the remaining field and inject a definitive, malicious DNS response that is accepted by the resolver.

3 TuDOOR ATTACK

3.1 TuDoor Attack Overview

The **TuDoor attack**, presented at the 2024 IEEE Symposium on Security and Privacy, represents a significant advancement in the field of DNS security vulnerabilities. Unlike previous attacks that primarily exploited protocol weaknesses or network side-channels (ICMP rate limiting), Tudoor targets **logic vulnerabilities** in the DNS response pre-processing mechanisms of recursive resolvers.

DNS response pre-processing is the phase where the resolver receives a packet, parses it, and validates it before extracting the answer to cache or forward it. Through the analyses of 28 different DNS software implementations, researchers discovered that while the fundamental rules of DNS are simple, the actual implementations are complex and often inconsistent in handling malformed packets.

The name "TuDoor" refers to a logic flaw that acts as a "hidden door" allowing attackers to inject malformed packets that should be rejected but are instead processed incorrectly, triggering specific state transitions. These vulnerabilities enable three types of attacks: Denial of Service (DoS), Ressource Consumption, and the most critical one that we will focus on in this paper, **Cache Poisoning**.

3.2 Mecanism of the Vulnerability

The core of the TuDoor cache poisoning attack relies on a specific logic vulnerability labeled V_{CP} . This vulnerability was prominently found in **Microsoft DNS**, a widely used enterprise resolver.

In a standard, secure implementation, when a recursive resolver opens a UDP socket to send a DNS query, it expects to receive a DNS *response* packet in return. A valid respond must have the **QR** (Query/Response) flag set to 1, indicating that it is indeed a response. If a packet arrives with the QR flag set to 0, indicating it is not a response, the resolver should logically discard it.

However, the researchers found that Microsoft DNS fails to enforce this rule correctly during the pre-processing phase. It allows a packet with the QR flag set to 0 to be received on the response socket, provided that the packet's "four-tuple" (source IP, source port, destination IP, destination port) matches an active query.

Crucially, when Microsoft DNS receives this query packet on its response socket, it does not simply discard it. Instead, it treats it as a new incoming question and initiates a recursive procedure to resolve the domain name contained within the malformed packet. This behavior creates a deterministic **side-channel**, and if the attacker sends a DNS query to a specific port on the resolver, and the resolver reacts by initiating a new resolution, the attacker is aware and can confirm that the port is open and active.

3.3 The DNS Cache Poisoning Attack Procedure

Leveraging the V_{CP} vulnerability, the TuDoor cache poisoning allows an off-path attacker to poison the DNS cache of a vulnerable resolver without relying on global ICMP rate limits or other side-channels. The attack attack proceeds in the following steps:

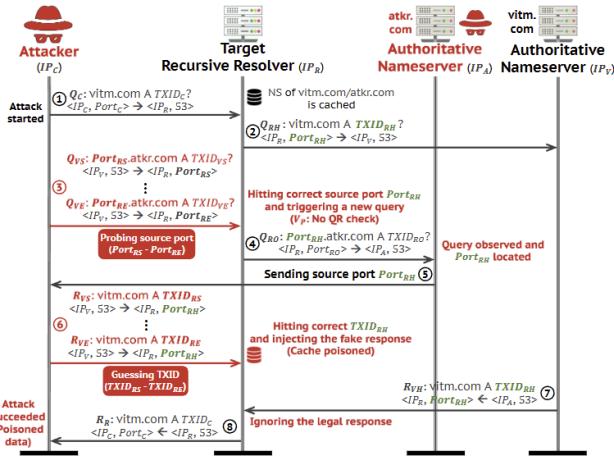


Fig. 2: Attack steps of DNSPOISONING.

- 1) **Setup and Trigger:** The attacker controls a nameserver (NS_{Atk}) for the domain `atkr.com`. The attacker initiates a DNS query to the target resolver for a victim domain (e.g., `vitm.com`).
- 2) **Target State:** The target resolver sends a legitimate query to the authoritative nameserver for `vitm.com`. It opens a randomized source port ($Port_{RH}$) and waits for the response.
- 3) **Probing Phase:** The attacker sends a series of spoofed DNS query packets to the target resolver, each with the QR flag set to 0. These packets are sent to different ports on the resolver, attempting to match the active source port ($Port_{RH}$). These queries ask the resolver to resolve a subdomain controlled by the attacker, encoding the guessed port number in the subdomain (e.g., `guess123.atkr.com`).
- 4) **Hitting the Door:** When a spoofed packet hits the closed port, it is ignored. However, when the packet hits the correct active port ($Port_{RH}$), the logic vulnerability V_{CP} is triggered. The resolver accepts the spoofed query packet.
- 5) **The Oracle Signal:** Upon accepting the query `port12345.atkr.com`, the target resolver immediately sends a new DNS query to the attacker's nameserver (NS_{Atk}) to resolve it.
- 6) **Port Identification:** The attacker observes the incoming query for `port12345.atkr.com` on their own nameserver. This confirms that the open port on the resolver is indeed 12345.
- 7) **Cache Poisoning:** Now possessing the correct source port, the attacker only need to brute-force the 16-bit $TxID$. The attacker sends a flood of spoofed DNS response packets to the resolver, each with different $TxID$ values, containing a malicious answer that redirects `vitm.com` to a server under the attacker's control.

3.4 Comparative Analysis and Impact

The TuDoor attack represents a significant evolution in DNS cache poisoning techniques compared to previous methods like Kaminsky and SAD DNS. The most notable feature of TuDoor is its speed. While Sad DNS relies on ICMP rate limits that can take minutes to exploit, Tudoor is highly efficient.

- **Deterministic vs Probabilistic:** Sad DNS infers port based on the probability of ICMP packet suppression. Tudoor receives a direct, deterministic signal from the resolver when the correct port is hit.
- **Execution Time:** Experimental results demonstrate that TuDoor can successfully identify the source port and poison the cache in **less than 1 second** (average 425ms for Microsoft DNS), whereas SAD DNS typically requires several minutes.

The implications of the TuDoor attack are profound. The researchers conducted internet-wide measurements and found that the vulnerability is widespread. It affects approximately 11.2% of open DNS resolvers and expose them to instant cache poisoning attacks.

4 DISCUSSION AND CONCLUSION

4.1 Impact on DNS Security

Duis aliquet dui in est. Donec eget est. Nunc lectus odio, varius at, fermentum in, accumsan non, enim. Aliquam erat volutpat. Proin sit amet nulla ut eros consectetur cursus. Phasellus dapibus aliquam justo. Nunc laoreet. Donec consequat placerat magna. Duis pretium tincidunt justo. Sed sollicitudin vestibulum quam. Nam quis ligula. Vivamus at metus. Etiam imperdiet imperdiet pede. Aenean turpis. Fusce augue velit, scelerisque sollicitudin, dictum vitae, tempor et, pede. Donec wisi sapien, feugiat in, fermentum ut, sollicitudin adipiscere, metus.

4.2 Causes and Mitigations

Donec vel nibh ut felis consectetur laoreet. Donec pede. Sed id quam id wisi laoreet suscipit. Nulla lectus dolor, aliquam ac, fringilla eget, mollis ut, orci. In pellentesque justo in ligula. Maecenas turpis. Donec eleifend leo at felis tincidunt consequat. Aenean turpis metus, malesuada sed, condimentum sit amet, auctor a, wisi. Pellentesque sapien elit, bibendum ac, posuere et, congue eu, felis. Vestibulum mattis libero quis metus scelerisque ultrices. Sed purus.

4.3 Conclusion

Donec molestie, magna ut luctus ultrices, tellus arcu nonummy velit, sit amet pulvinar elit justo et mauris. In pede. Maecenas euismod elit eu erat. Aliquam augue wisi, facilisis congue, suscipit in, adipiscing et, ante. In justo. Cras lobortis neque ac ipsum. Nunc fermentum massa at ante. Donec orci tortor, egestas sit amet, ultrices eget, venenatis eget, mi. Maecenas vehicula leo semper est. Mauris vel metus. Aliquam erat volutpat. In rhoncus sapien ac tellus. Pellentesque ligula.

REFERENCES

- [1] X. Li *et al.*, “TuDoor Attack: Systematically Exploring and Exploiting Logic Vulnerabilities in DNS Response Pre-processing with Malformed Packets,” 2024 IEEE Symposium on Security and Privacy (SP), pp. 4459-4477, 2024.
- [2] D. Kaminsky, “DNS Vulnerability,” Black Hat USA, 2008.
- [3] K. Qian *et al.*, “SAD DNS: Exploiting Weakened Trust in DNS,” ACM CCS, 2020.
- [4] H. Kopka and P. W. Daly, *A Guide to L^AT_EX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.