Report: Cache Poisoning Dnsmasq without Advanced technilogy only via one character (Severity: Critical)

By Fasheng Miao from Tsinghua University and Xiang Li [1] from AOSP Laboratory of Nankai University.

On May 10, 2025

Overview

We discovered a new logic vulnerability in the Dnsmasq software. Attackers can inject arbitrary malicious resource records to **poison the domain names**. The effects of such an exploitation would be widespread and extremely impactful, as **attackers do not need any advanced techniques—just brute-force using the birthday paradox within an acceptable amount of time**. We named this attack **Shar Attack**. We conducted 20 attack attempts, all of which succeeded, with an average execution time of **9,469 seconds**.

Moreover, **Shar Attack** can further **facilitate well-known cache poisoning attacks** against Dnsmasq software, such as **SADDNS**[4] and **Tudoor**[5].

Vulnerability Details

After analyzing the source code of various DNS software and testing their support for special characters, we turned our attention to the caching mechanism. If an authoritative nameserver fails to respond to a resolver's query, the resolver may wait or resend the same query multiple times, lasting for over 5 seconds.

> **Note:** Special characters refer to the 66 ASCII characters excluding letters('a'-'z','A'-'z') and digits('0'-'9'). These special characters include both printable symbols and non-printable control characters, as defined by their ASCII codes.

Specifically, we have discovered a new vulnerability in the Dnsmasq DNS Software (Dnsmasq) across all its versions, which enables the attacker to inject arbitrary malicious resource records and **poison the domain names**.

The vulnerability is shown below.

After receiving a query (e.g., `test.example.com`) from the client (*step 1*), **Dnsmasq** forwards the request to an upstream **recursive resolver** (*step 2*). Under normal conditions, the resolver returns either a response with correct answers or an `NXDomain`/`SERVFAIL` reply (*step 3*). Dnsmasq then relays this response back to the client (*step 4*).

However, if the upstream recursive resolver does **not** respond to Dnsmasq (*step 5*), Dnsmasq simply waits silently and performs no validation or detection against potential injection attempts. This creates a significant **attack window** that can be exploited by attackers.

> **Remark 1:** The query can target any arbitrary domain name, not just those within a specific zone.

> **Remark 2:** Many recursive resolvers directly discard queries containing certain characters (e.g., `~`, `!`, `*`, `_`).

> **Remark 3:** If the recursive resolver remains silent, Dnsmasq also stays silent to its downstream clients, exposing them to the **SHAR**

The vulnerability is that Dnsmasq should not simply wait for a reply and does not perform any detection against injection attacks.

This vulnerability provides a large attack window. During this period, attackers can probe certain ports and brute-force all 65,536 transaction ID (TxID) values in each round. If source port and TxID are matched, this attack will succeed. This attack is guaranteed to succeed as the number of rounds increases.

Cache Poisoning Attack without Advanced technilogy only via one character

We propose a novel cache poisoning technique targeting **Dnsmasq DNS software**, named the **SHAR Attack**, which is both supremely effective and impactful, yet requires **only a single character modification** to launch. Unlike traditional attacks, **SHAR Attack** does not require advanced capabilities such as IP fragmentation, query aggregation, or side-channel timing inference.

SHAR exploits **inconsistencies in the handling of special characters** (e.g., `~`, `!`, `*`, `_`) across different DNS components and implementations. By injecting a single crafted character into the domain name, the attacker induces **silence** from upstream recursive resolvers, avoiding NXDOMAIN or SERVFAIL responses. This silent behavior **extends the attack window**, enabling successful cache poisoning with higher probability and reduced complexity.

We detail two variants of this attack:

Variant 1: V_{AC} (Enhancing Classic Cache Poisoning Attacks)

Goal: Extend the time window for spoofed responses to arrive before legitimate ones in off-path attacks.

Mechanism:

- Inject a special character (e.g., `~`, `!`, `*`, `_`) into a query.
- This causes the resolver (e.g., Dnsmasq) to forward a malformed query upstream.
- Some upstream recursive resolvers **remain silent** (do not respond at all).
- The forwarder retains the pending query state without receiving a reply.

Impact:

- The attacker can spoof a response **without racing** the legitimate one.
- Dramatically increases success rate of classic cache poisoning techniques.
- Does not modify the underlying algorithm, only adds a special character.

Variant 2: V_{BF} (Feasibility of Brute-Forcing TxID and Source Port)

Goal: Enable brute-force poisoning of resolvers even in presence of source port randomization and TxID defense.

Mechanism:

- Leverages the same one-character injection as V_{AC}.
- Ensures no response is returned from the upstream recursive resolvers.
- The forwarder **keeps retrying or waits indefinitely**, giving attacker **unlimited attempts** to guess TxID and source port.

Impact:

- Overcome 16-bit TxID and 16-bit port randomization protections (`2^32` search space).
- No side-channel, packet fragmentation, or DNS reDnsmasqing required.
- Exposes the fundamental insecurity in DNS protocol parsing logic.

Attack Steps

Before the attack, we assume that the attacker can initiate a domain name query request containing special characters, which can be easily achieved by directly encoding the domain name into the DNS packet. Additionally, the attacker needs to analyze which special characters the resolver can forward and which special characters the victim SLD nameserver does not support and directly discards without responding to the resolver, and use these characters as the target characters. The preparation steps are as follows:

1. The attacker directly encodes the domain name containing special characters into the DNS packet.
2. Initiate a query for `<specialchar>.attacker.com` to the victim forwarder and analyze which special characters the forwarder can query.

   > **Note:** Dnsmasq is capable of querying 65 special characters within domain names, excluding the dot (.) as a standalone label.

3. Initiate a query for `<specialchar>.viticm.com` to the victim SLD and analyze which characters in the query will be discarded by the upstream recursive resolvers(`..example.com` can't be forwarded by Dnsmasq).

   > **Note:** For unsupported characters, the upstream recursive resolvers may silently drop the query without sending any response to the resolver.

The attack steps are:

1. **Craft and Send Trigger Queries**
   The attacker sends DNS queries of the form, `<targetchar>.<random_string>.viticm.com`.

   - `<targetchar>` is a specially crafted character (e.g., `!`, `-`, `\`) designed to **trigger silence** from upstream recursive resolvers.
   - `<random_string>` is a random string to **bypass local caching**.

2. **Resolver Forwards Query**
   The victim forwarder forwards the query to upstream recursive resolvers. For each forwarded query:

   - A **random TxID** and **random source port** are assigned (typical 16-bit each).
   - If the upstream nameserver fails to process the malformed query, it **remains silent**.

3. **Extended Attack Window via Silence**
   The upstream silence **prevents legitimate responses** (e.g., NXDomain/ServFail), so the resolver

continues to **wait**.

- This **extends the time window** available for the attacker to spoof a valid response.

4. **Brute-Force Spoofing**
   Within the extended window, the attacker **injects spoofed DNS responses** toward the victim resolver, attempting to:
   - Guess the correct **TxID** (16 bits).
   - Guess the correct **source port** (typically guessing 5 ports per round).
   - Each round probes 65,536 TxIDs × 5 ports.
5. **Verification and Retry**
   After spoofing, the attacker checks the DNS response (via a controlled domain):
   - If it resolves to the expected **sentinel IP (e.g., a.t.k.r)**, poisoning is **successful**.
   - If it resolves to the **fallback domain (e.g., v.c.t.m)**, poisoning **failed**, and the process restarts.

> **Note (Birthday Paradox):**
> The probability of success increases with rounds due to the **Birthday Paradox**. The cumulative success probability after x rounds is:
> $P_{\text{success}} = 1 - \left(1 - \frac{5}{65536} \right)^x$

Experimental Results

We performed 20 attack runs against **Dnsmasq** All attacks succeeded.

Software	Avg. Rounds	Avg. Time (s)	Success Rate
Dnsmasq	7,947	9,469	20 / 20

- **Shortest poisoning time**:
- Dnsmasq: 193s
- **Longest poisoning time**:
- Dnsmasq: 21,903s
- **Peak outgoing bandwidth**: 297 Mbps

> Note: The V_{BF} variant of the **SHAR attack** proves that **off-path cache poisoning is once again feasible**. This undermines longstanding assumptions about DNS security since 2008, particularly in resolvers like Dnsmasq which assume upstream recursive resolvers silence is benign.

> Note: We can further accelerate the guessing process by increasing the number of ports guessed in each attempt. (In our experiments conducted on an ordinary laptop, we guessed 5 ports per attempt. Doubling the number of guessed ports halves the required time.)

> Note:

Threat Surface

Software	Version

Software	Version
Dnsmasq	All versions are vulnerable

PoC

We registered the domain name `viticm.com` in the real-world Internet environment and configured the upstream recursive resolvers: `resolver1`. By leveraging a single specially crafted character, we were able to induce silence from `resolver1`. Because for this character, `resolver1` directly discard the query without response.

To validate this attack, we can simulate whether to reply to DNS packets by shutting down or starting `resolver1`.

Mitigation

The forwarders can enhance security by blocking malicious response injections through anomaly detection and rate limiting techniques, as implemented in PowerDNS [6].

Reference

[1] Xiang Li's homepage: https://lixiang521.com/.

[2] RFC1034: https://datatracker.ietf.org/doc/html/rfc1034.

[3] RFC2182: https://datatracker.ietf.org/doc/html/rfc2182.

[4] SADDNS: https://www.saddns.net/

[5] Tudoor：https://tudoor.net/

[6] PowerDNS: https://docs.powerdns.com/recursor/settings.html#spoof-nearmiss-max