

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

DNS Cache Poisoning Like it's 2006

Omer Ben-Simhon and Amit Klein, *Hebrew University of Jerusalem*

https://www.usenix.org/conference/usenixsecurity26/presentation/ben-simhon

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology



USENIX Security '26 Artifact Appendix: DNS Cache Poisoning Like it's 2006

Omer Ben-Simhon
Hebrew University of Jerusalem

Amit Klein
Hebrew University of Jerusalem

A Artifact Appendix

A.1 Abstract

This artifact accompanies the paper *"DNS Cache Poisoning Like it's 2006"* and demonstrates practical DNS cache poisoning attacks against vulnerable versions of the BIND 9 recursive resolver. The artifact implements two attack variants—*RRset* and *RRset-ANY*—and explains how to build a complete, self-contained laboratory environment for reproducing the paper's core claims. Using a controlled four-host DNS setup, the artifact allows evaluators to reproduce successful cache poisoning under realistic resolver configurations. The testing instructions in the artifact are designed to reproduce the results described in Section 5.2 of the paper ("Baseline experiments"), and specifically the RRset and RRset-ANY tests against BIND 9.20 (see Table 2 in the paper). Other experiments are not explicitly included, though a knowledgeable tester can probably adapt the setup to test additional scenarios with some effort.

A.2 Description & Requirements

This artifact demonstrates practical DNS cache poisoning attacks against vulnerable BIND 9 recursive resolvers in a controlled laboratory environment. The attack is orchestrated through the interaction of four distinct hosts, each playing a well-defined role in the DNS resolution process and the attack workflow. The artifact does not recreate the exact Azure cloud environment used in the paper's large-scale experiments. Instead, it reproduces the same attack logic and outcomes in a controlled local (LAN or virtualized) setup. In Azure, cloud anti-spoofing restrictions required us to use the "spoofing-at-the-destination" technique as an environmental workaround. This does not change the attack itself, but only how spoofed packets are injected under cloud constraints. The artifact recommends a local setup because it avoids these cloud-specific restrictions and simplifies reproduction, while preserving the core attack behavior, resolver interactions, and success conditions described in the paper.

System architecture. The experimental setup consists of the following four machines (we denote each machine with a bracketed, gothic, colored letter here, and later use this notation when prescribing shell commands, to clarify in which machine(s) they should run):

- **Resolver (BIND)**: a BIND 9 recursive resolver that serves DNS queries from clients and maintains a cache. This resolver is the target of the cache poisoning attack. Denoted by $\langle\mathcal{R}\rangle$.
- **Legitimate Authoritative Name Server (ANS)**: an authoritative DNS server for the domain `victim.example`, which serves legitimate DNS records – this is the domain the attacker targets. Denoted by $\langle\mathcal{V}\rangle$.
- **Attacker Authoritative Name Server (ANS)**: an authoritative DNS server for the domain `attacker.example`, controlled by the attacker and used to provide malicious resource records during the attack. Denoted by $\langle\mathcal{A}\rangle$.
- **Attacker Client**: an attacker client machine that triggers DNS resolution at the resolver and launches the cache poisoning attack by sending legitimate queries alongside spoofed DNS responses. Denoted by $\langle\mathcal{C}\rangle$.

All experiments are conducted in an isolated network under the evaluator's control.

Poisoning target. The concrete poisoning target in the artifact is the DNS name:

`www0.victim.example`

Prior to the attack, this name (more accurately, its DNS A record) is legitimately resolved by the resolver to an IPv4 address served by the legitimate authoritative server:

`www0.victim.example` → `198.51.100.10`

The goal of the attack is to poison the resolver's cache such that subsequent queries for this name are resolved to an attacker-chosen IPv4 address:

`www0.victim.example` → `198.51.100.66`

The artifact demonstrates this by first seeding the resolver cache with the genuine record and then racing spoofed responses against the legitimate response from the legitimate

authoritative server, as described in the paper. Successful poisoning is verified by querying the resolver after the attack, and observing that it returns the attacker-controlled address from its cache.

There are two hard-wired poisoned answers involved. The first one (with TTL=35s) is the "Kaminsky answer" (sent by the attacker with the predicted TXID and UDP port) which contains a malicious NS record (+glue) for `www0`, pointing at the attacker client IP address. The second poisoned answer (with TTL=30s) is the "payload answer" generated for the `A?www0` query sent by the BIND9 resolver to the attacker, once the A record for `www0` expires from BIND9's cache. The poisoned IPv4 address (`198.51.100.66`) used in this answer is hard-coded in the script for reproducibility. In real attacks, attackers would typically use a much larger TTL value in the payload answer, to maximize persistence.

A.2.1 Security, privacy, and ethical concerns

This artifact sends *spoofed UDP DNS packets* and deliberately races legitimate DNS responses. Running the artifact on production networks or public infrastructure may violate acceptable-use policies and local regulations and laws. Evaluators **must** execute the artifact only on isolated laboratory networks (physical or virtual) they fully own and fully under their control. The artifact does not exfiltrate data, access third-party systems, or perform persistence beyond temporary DNS cache entries.

A.2.2 How to access

The artifact is archived on Zenodo. Once the paper is published, the artifact will be accessible via the following stable reference:

`https://doi.org/10.5281/zenodo.17762025`

A.2.3 Hardware dependencies

The artifact runs on commodity x86-64 hardware. Evaluators may use either physical machines or virtual machines.

The attack requires **spoofed UDP packets** to reach the resolver. Many ISPs and cloud providers (e.g., public IaaS platforms such as Azure) block spoofed packets, even on internal virtual networks. **Strong recommendation:** perform all testing on a physical LAN, or a locally virtualized LAN (e.g., VMware, VirtualBox, or QEMU).

In addition, don't use networks that have NATs or any other functionality that may alter packets. For example, don't use VMware's NAT mode; you should use e.g. bridged mode instead.

For consistency with the experimental results reported in the paper, we strongly recommend using **Ubuntu 24.04** on all machines and using **BIND 9** as the authoritative name server software on the legitimate authoritative server and the attacker authoritative server machines.

BIND versions. Only vulnerable BIND 9 versions can reproduce the attacks.
Vulnerable (usable) versions: BIND 9.18.0–9.18.39, BIND 9.20.0–9.20.13, BIND 9.21.0–9.21.12.
Patched versions (not vulnerable; do not use): BIND 9.18.41+, BIND 9.20.15+, BIND 9.21.14+.
The recommended version for evaluation is **BIND 9.20.13**. As far as we know, from the perspective of the vulnerabilities described in the paper, BIND 9.20.13 behaves identically to BIND 9.20.11 (which is the version paper baseline experiments use). Instructions for downloading and building this BIND version are provided in subsubsection A.3.1.

System packages. The following packages are required on all machines: **build-essential, gcc, g++-10, make, python3 (3.10+), python3-pip, python3-venv, python3-numpy, python3-dns, python3-dnspython, python3-scapy, libssl-dev, wget, curl, git, net-tools, dnsutils, tcptraceroute**.

Additional build dependencies are required on machines running BIND 9 (required for all of the machines except the attacker client): **libuv1-dev, libnghttp2-dev, libcap-dev, libjson-c-dev, pkg-config, autoconf, automake, libtool, liburcu-dev**.

A.2.4 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

After installing the required packages for each machine as described in previous section, proceed to the instruction in this section.

Installing BIND 9.20.13 from source. ⟨$\mathcal{A}$⟩ ⟨$\mathcal{V}$⟩ ⟨$\mathcal{R}$⟩
BIND 9.20.13 can be obtained from the ISC archive: `https://ftp.isc.org/isc/bind9/9.20.13/bind-9.20.13.tar.xz`
Assuming BIND 9 was downloaded to to a file named `bind-9.20.13.tar.xz`, unpack, build, and install it:

```
tar -xf bind-9.20.13.tar.xz
cd bind-9.20.13
./configure
make
sudo make install
sudo ldconfig -v
sudo chmod 777 .
```

System configuration. ⟨$\mathcal{A}$⟩ ⟨$\mathcal{V}$⟩ ⟨$\mathcal{R}$⟩ The built-in Linux stub resolver must be disabled on all machines:

```
sudo systemctl stop systemd-resolved
sudo systemctl disable systemd-resolved
```

⟨$\mathcal{R}$⟩ Between experiments, the resolver must be restarted to clear cached state. If running `named` from source (as suggested above) run:

```
sudo killall named
```

Configuration files and IPv4 address placeholders. All configuration files provided in the artifact contain placeholder IP addresses that **must be replaced** with their corresponding concrete IPv4 addresses before running the experiments: `<resolver_ip>`, `<legitimate_ans_ip>`, `<attacker_ans_ip>`, `<attacker_client_ip>`.

Machine-specific configuration.

Configuration template (DNS servers). For each DNS server (authoritative or resolver), the setup follows the same procedure: (i) copy the relevant configuration files to `/etc`, (ii) replace all placeholder IP addresses, and (iii) start `named` with the corresponding main configuration file.

Attacker Authoritative Name Server (ANS): ⟨$\mathcal{A}$⟩ Configuration files: `attacker.named.conf`, `attacker.named.conf.local`, `attacker.example.zone`. Note that the attacker ANS IP address appears twice and must be updated consistently. Start the server using:

```
sudo named -4 -c /etc/attacker.named.conf -g
```

Legitimate Authoritative Name Server (ANS): ⟨$\mathcal{V}$⟩ Configuration files: `legitimate.named.conf`, `legitimate.named.conf.options`, `legitimate.named.conf.local`, `victim.example.zone`. Start the server using:

```
sudo named -4 -c /etc/legitimate.named.conf -g
```

Resolver (BIND): ⟨$\mathcal{R}$⟩ Configuration files: `resolver.named.conf`, `resolver.named.conf.options`, `resolver.named.conf.local`, `resolver.example.zone`. Start the resolver using:

```
sudo named -4 -c /etc/resolver.named.conf -g
```

Attacker Client: ⟨$\mathcal{C}$⟩ All artifact files should be copied to this machine. Build the attacker's script components using:

```
make all
```

⟨$\mathcal{C}$⟩ Run the attack script with the appropriate arguments, e.g.:

```
sudo python3 attacker_client.py ...
```

(Concrete command lines are given in subsubsection A.4.2)

Network timing configuration. To create a reproducible attack window on low-latency networks, evaluators are recommended to delay responses from the legitimate authoritative server: ⟨$\mathcal{V}$⟩

```
sudo tc qdisc add dev <interface_name> root netem
↪ delay 80ms
```

Where `<interface_name>` is the interface used to send traffic to the other hosts.

After completing the experiments, the delay must be removed: ⟨$\mathcal{V}$⟩

```
sudo tc qdisc del dev <interface_name> root netem
```

A.3.2 Basic Test

⟨$\mathcal{R}$⟩ Before running the attack, evaluators should verify that the resolver uses the expected ephemeral port range:

```
sysctl net.ipv4.ip_local_port_range
```

which should return `32768 60999`.

⟨$\mathcal{C}$⟩ To verify correct installation, evaluators should query the resolver before launching the attack:

```
dig @<resolver_ip> A www0.victim.example
```

Expected output (before attack):

```
www0.victim.example.  <TTL>  IN  A  198.51.100.10
```

This confirms that the resolver correctly fetches and caches the legitimate record.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *A remote attacker can poison the cache of a vulnerable BIND 9 recursive resolver using an RRset-based attack over UDP or TCP.* This claim is demonstrated by experiment (E1) and corresponds to the RRset attack described in Section 4.2.1 of the paper and evaluated in Section 5.2.

(C2): *A remote attacker can poison the cache of a vulnerable BIND 9 recursive resolver using the RRset-ANY attack variant over both UDP and TCP.* This claim is demonstrated by experiment (E2) and corresponds to the RRset-ANY attack described in Section 4.2.1 of the paper and evaluated in Section 5.2.

A.4.2 Experiments

RRset (E1) / RRset-ANY (E2) Attacks, over TCP / UDP [1 human-minute per each combination]

Herein we describe four experiments (combinations): Two E1 combinations (RRset over TCP and RRset over UDP), and two E2 combinations (RRset-ANY over TCP and RRset-ANY over UDP).

Preparation: Ensure that all four machines are running and configured as described above in subsubsection A.3.1.

Execution: ⟨C⟩ On the attacker client machine, from within the artifact directory, run the following command template, setting the parameters as follows:

- **`-mode`** to `rrset` or `rrset_any`
- **`-protocol`** to `tcp` or `udp`
- **`-rrset_domain`** to `x.attacker.example` (RRset), or to `x1.attacker.example` / `x2.attacker.example` for RRset-ANY over TCP / UDP, respectively

```
sudo python3 attacker_client.py \
  --mode <rrset|rrset_any> \
  --protocol <tcp|udp> \
  --iterations 1 \
  --resolver_ip <resolver_ip> \
  --attacker_ans_ip <attacker_ans_ip> \
  --attacker_ans_domain attacker.example \
  --victim_ans_domain victim.example \
  --legitimate_ans_ip <legitimate_ans_ip> \
  --logging console \
  --rrset_domain <x|x1|x2>.attacker.example \
  --public_ip <attacker_client_ip>
```

The script automatically seeds the resolver cache with the legitimate record, executes the attack, and terminates after approximately 35 seconds.

Results: During execution, the script prints the message:

```
Attack Succeeded!
```

⟨C⟩ Immediately after the script finishes, query the resolver:

```
dig @<resolver_ip> -4 A www0.victim.example
```

The expected response is a poisoned cache entry:

```
www0.victim.example.   <some_TTL>   IN   A
↪  198.51.100.66
```

A.5 Notes on Reusability

This artifact can be reused to evaluate:

- Different BIND 9 versions and configurations
- TCP vs. UDP transport effects
- Defensive mechanisms (e.g., patched BIND versions)

The modular configuration files allow easy adaptation to additional experimental scenarios.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.