USENIX Security '25 Artifact Appendix: Transparent Attested DNS for Confidential Computing Services

Antoine Delignat-Lavaud
Azure Research, Microsoft

Cédric Fournet
Azure Research, Microsoft

Kapil Vaswani
Azure Research, Microsoft

Manuel Costa
Azure Research, Microsoft

Sylvan Clebsch
Azure Research, Microsoft

Christoph M. Wintersteiger
Imandra

A Artifact Appendix

A.1 Abstract

Our artifacts consist mainly of an implementation of an authoritative DNSSEC server that implements the attested DNS protocol, as described in the paper, and a browser extension that implements the client protocol.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Our implementation of attested DNS is for research purposes only. We do not recommend directly exposing our attested DNS server to the public Internet, as it lacks some of the recommended defenses against classical DNS-based attacks, in particular DNS amplification attacks over UDP.

If you want to test our server over the public Internet (for instance, to test client latency or the browser extension), consider exposing it through Bind using a forwarder zone. We provide a sample Bind configuration, see the README file for details.

A.2.2 How to access

The version of attested DNS used in this paper is available from https://doi.org/10.5281/zenodo.15611255. It is based on the open source repository https://github.com/microsoft/ccfdns - however, please note that the open source repository has significantly diverged from the version described in the paper. While the open source version is significantly more up to date, it is missing several features that are evaluated in the paper.

A.2.3 Hardware dependencies

Our artifacts requires Intel SGX capable hardware and must run with DCAP (Data Center Attestation Primitives) attestation. EPID-based attestation is not supported. We have tested our code on Azure `Standard_DC*ds_v3` VMs, other cloud provides are expected to work but may require additional OpenEnclave configuration steps that we do not provide. Refer to the OpenEnclave manual for details on other platforms (including non-cloud based setups and simulation mode).

A.2.4 Software dependencies

This version of attested DNS has been tested on Ubuntu 20.04. Our main dependencies are OpenEnclave (https://github.com/microsoft/openenclave), and the Confidential Consortium Framework (https://github.com/microsoft/CCF), but our artifact package includes compatible versions. We use DNSPyre (https://github.com/Tantalor93/dnspyre) for benchmarking our server.

A.2.5 Benchmarks

None.

A.3 Set-up

We assume you have downloaded the artifact package and extracted it to your home directory `~/adns`.

A.3.1 Installation

We have automated the installation of dependencies using an Ansible playbook. You can run the playbook with the following commands:

```
cd ~/adns/CCF/getting_started/setup_vm
./run.sh app-dev.yml
```

This script will also install Ansible and its dependencies.

The next step is to install CCF:

```
cd ~/adns/CCF
sudo apt install ./ccf_sgx_5.0.0_dev12_amd64.deb
```

To build the aDNS server, you will also need LLVM 18, which you can install with:

```
sudo apt install clang-18
```

The next step is to build the aDNS enclave:

```
cd ~/adns/ccfdns
./build.sh
```

A.3.2 Basic Test

The main tool for testing our aDNS server is a script that starts an instance of aDNS on port 5353 configured for the zone `service.confidential`. The script also registers two partifipating TEEs under `sgx.service.confidential` and `sev-snp.service.confidential`. To simplify the evaluation and avoid multiple hardware dependencies, these are not live services but use pre-recorded attestation reports.

```
cd ~/adns/scripts
./run.sh
```

The process of starting the server involves the registration of the two TEEs, the DNSSEC signature of the zone, and the issuance of ACME certificates.

```
Testing SGX service registration...
...
SGX service registration completed in 176.34 ms
Testing registration policy update by consortium member
...
New policy set in 249.15 ms
Testing SEV-SNP service registration
...
SEV-SNP service registration completed in 156.80 ms
Testing re-signature of zone
...
Zone re-signature completed in 14.24 ms
Testing ACME certificate issuance through aDNS
ACME certificate returned: {"certificate":
  "-----BEGIN CERTIFICATE-----
 ....-----END CERTIFICATE-----\n"}
Certificate returned in 2.01 ms
```

The server will keep running until interrupted (stop it with `ctrl+c`). Please note that the experiments below require the server to be running. You can only run one single instance of the server at a time.

For testing aDNS clients, note that the aDNS server is itself a registered TEE of the zone, available at `https://ns1.service.confidential:8443`. Note that all ACME certificates use a local root CA in `~/adns/scripts/pebble.pem`. You HTTPS client should be configured to use that root CA when connecting to aDNS. For instance, to see all the records configured in aDNS's zone, run the following command:

```
curl --cacert pebble.pem \
  https://ns1.service.confidential:8443/app/dump
```

We need to install DNSPyre to measure the performance of the aDNS zone:

```
sudo apt install \
  ~/adns/dnspyre_3.5.1_linux_amd64.deb
```

Next, we setup a local installation of the `bind` DNS server. This will be used later to measure the scalability of aDNS through Bind9, using its highly optimized cache.

```
cd ~/adns
sudo apt install bind9
sudo cp named.conf.* /etc/bind/
sudo service named restart
```

The packaged Bind configuration file will forward queries to `service.confidential` to port 5353. You can check your setup is correct with dig:

```
dig @127.0.0.1 _8443._tcp.ns1.service.confidential TLSA
```

If aDNS is running and well configured you should see the DANE record:

```
;; ANSWER SECTION:
_8443._tcp.ns1.service.confidential. 1440 IN TLSA
    3 1 0 3076301006072A8648CE3D020106052B81...
```

Finally, to help test legacy client, we recommend you set the local bind as your system resolver.

```
sudo cp resolved.conf /etc/systemd/
sudo service systemd-resolved restart
```

We will use the Firefox extension to measure client latency using the browser networking profiler. If you have configured bind as your system resolver, you can connect to `service.confidential` directly, but if you plan to connect from a remote client, you may need to setup a DoH resolver with a custom certificate, which is not recommended to do on a personal device.

```
sudo apt install tightvncserver firefox
```

To setup the extension, we recommend `web-ext`, which is available on the Node Package Manager. Make sure Firefox and Node.js are installed, for instance on Debian systems:

```
curl -fsSL https://deb.nodesource.com/setup_23.x \
  -o nodesource_setup.sh
sudo -E bash nodesource_setup.sh
sudo apt-get install -y nodejs
sudo npm install -g web-ext
```

Then from the extension directory, call `web-ext` to start a browser session with the extension installed:

```
cd adns/extension
web-ext run
```

A.4 Evaluation workflow

A.4.1 Major Claims

We make two main claims in the paper

(C1): The impact of aDNS on the DNS serving infrastructure is at worst 50%, that is, going from clients that only check the A record of a service to clients that verify DNSSEC and request the full attestation evidence from DNS (worst case scenario) only impacts the throughput of the authoritative server by about half.

(C2): aDNS clients that verify attestation at the same time as the TLS handshake do not observe any additional latency for typical client roundtrip times of about 25ms.

A.4.2 Experiments

(E1): *[Server scalability] [40 human-minutes]* We evaluate the latency/throughput by submitting an increasing amount of requests with dnspyre and measuring the P99 latency.

Preparation: Start the aDNS server with `scripts/run.sh` and make sure your local bind server is well configured.

Execution: We use the script `scripts/latency.sh`, which accepts 4 arguments: the type of client to simulate (can be `basic`, `dane` or `adns`), the DNS server IP (`127.0.0.1` by default), its port (`5353` by default) and the number of queries in each measurement (by default 10000). For each 3 type of client, we run the script and collect the reported `Questions per second` and `p99` latency results. We first do this by accessing our aDNS server directly (on port 5353), then through bind (on port 53):

```
./latency.sh basic
./latency.sh dane
./latency.sh adns
./latency.sh basic 127.0.0.1 53
./latency.sh dane 127.0.0.1 53
./latency.sh adns 127.0.0.1 53
```

Results: The results should roughly match the results of Figure 4 and Figure 5. As we increase the number of concurrent requests, the throughput and P99 latency gradually increase. Bind is about 2 orders of magnitude better than our own implementation. More importantly, the maximum throughput achieved by aDNS clients is about half that of basic clients.

(E2): *[Client latency] [5 human-minutes]*

Preparation: Start the aDNS server with `scripts/run.sh`. To keep things simple, we assume you will run Firefox on a machine configured to use the locally configured bind as a system resolver. For instance, use VNC to connect to the server where aDNS is running and launch a Firefox session with the extension using `web-ext run`.

Execution: Start the network profiler by going to the Network tab of the developer tools (F12). Click the stopwatch in the bottom left corner to start profiling. Navigate to `https://ns1.service.confidential:8443` and measure the total latency (until the HTTP request is sent). Disable the extension and repeat the experiment. You can check the extension developer console to see the details of the attestation verification happening in the background.

Results: Unless you have very low RTT to the aDNS server, the total latency should be the same with or without aDNS.

A.5 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/usenixsec2025/`.