ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

USENIX Security '25 Artifact Appendix: "STEK Sharing is Not Caring: Bypassing TLS Authentication in Web Servers using Session Tickets"

Sven Hebrok, *Paderborn University*
Tim Leonhard Storm, *Paderborn University*
Felix Matthias Cramer, *Paderborn University*
Maximilian Radoy, *Paderborn University*
Juraj Somorovsky, *Paderborn University*

August 6, 2025

A Artifact Appendix

A.1 Abstract

Our artifact consists of three separate software submissions mapping to the three major parts of our paper:

Artifact I A tool for offline scans evaluating the impact of different SNI and `Host` header combinations on server contents when using session resumption.

Artifact II A tool for offline scans evaluating the impact of different SNI and `Host` header combinations on TLS client authentication when using session resumption.

Artifact III A tool(chain) for online scans evaluating the impact on served content when using session tickets issued by related, but different hosts.

Artifact I and Artifact II can be used to fully reproduce our results mostly offline. Both only require internet access to download docker containers. Artifact III can be used to reproduce the *online*, but due to servers changing (partially in response to our disclosure), the results will likely differ. We provide a slimmed down version running against local Docker containers instead, which demonstrates the workflow and functionality of our software.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Artifacts I and II download and run Docker images from the official registry. All containers use the official images from the respective web server maintainers. We also create our own Docker containers based on the official Docker-in-Docker images to run our artifacts. These containers are run with the `--privileged` flag.

A.2.2 How to access

The artifacts are made permanently accessible on zenodo, available at

`https://doi.org/10.5281/zenodo.15474656`

A.2.3 Hardware dependencies

None.

A.2.4 Software dependencies

Because all three artifacts use radically different setups, we provide Docker images for Artifact I and Artifact II. Extended instructions are contained in the respective README.

Artifact I OpenSSL, Docker

Artifact II OpenSSL, Docker

Artifact III Python $\geq$ 3.12, Pip, Docker, Golang 1.20, cmake, jqo, libjudy, libgmp, libpcap, byacc, flex, libjson, libunistring, curl, git

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

The instructions shown here use the dockerized versions where possible, but instructions for running on baremetal are available in the README.

Artifact I Build the Docker container:

```
cd server-tests
docker build -t server-tests:latest .
```

Artifact II Build the Docker container:

```
cd Auth_Dotsche
docker build -t authdotsche:latest . -f
 ↪ Dockerfile
```

Artifact III *We strongly recommend using a VM, because we assume sudo for a lot of operations.* Build all required tools (including our respective forks):

```
git clone --recurse-submodules ...
sudo ./build_dependencies.sh
```

A.3.2 Basic Test

Artifact I Spawn a pre-configured `nginx` Docker container using

```
docker run --rm --name server-tests -it
 ↪ --privileged server-tests:latest deploy
 ↪ nginx one-server
```

`curl` commands for different SNIs are printed, that should result in different dummy websites being received. Execute these via

```
docker exec server-tests <curl cmd>
```

Artifact II Omitted because the complete test run is fully automated and should only take minutes on modern mid-range hardware. No reasonable subset of setups or checks would simplify the setup or shorten the runtime for a considerable amount.

Artifact III Check that the different binaries are built correctly:

```
./zdns/zdns --help
./zmap/src/zmap --help
./zmapv6/src/zmap --help
./zgrab2_tls13/cmd/zgrab2/zgrab2 --help
```

Also check whether the dummy servers and DNS resolution are working:

```
./setup.sh
dig @127.0.0.1 -p 8053 a.com
```

This should yield some local Docker IP.

A.4 Evaluation workflow

Because of the differences between our artifacts, we split this section into three parts.

A.4.1 Major Claims Artifact I

(C1): We claim that Table 1 presented in Section 4 is accurate for the versions listed in appendix A.

A.4.2 Experiments Artifact I

(E1): Generating software results *[2 human-minutes + 15 compute-minutes]*:
Preparation: Install the software according to appendix A.3.1.
Execution: Run to execute all testcases against all supported software configurations.

```
docker run --privileged -it -v ./out:/out
 ↪ server-tests:latest evaluate --outdir
 ↪ out
```

Optionally, you can restrict the output to a single software or testcase with `--software` and `--case` respectively. A full list of available options can be found in `testcases/configuration.yml` By default results are written in the working directory, but `--outdir` can be used to change the directory (Note that this refers to the directory inside the container for dockerized usage).
Results: The file results.{json, csv, jsonl} lists how each of the different softwares and their configurations react to different SNI and `Host` header, based on the virtual host issuing the session ticket and the receiving virtual host. This is the raw data for Table 1. Note that the table contains only the `one-server` case.
You can either verify the results against Table 1 by hand or refer to **(E2)**.

(E2): Verifying the table *[5 human-minutes + 1 compute-minute]*:
Preparation: Run (E1) in **full** configuration (without `--software` or `--case`).
Execution: Run

```
docker run --privileged -it -v ./out:/out
 ↪ server-tests:latest postprocess
 ↪ out/results.jsonl
```

and verify that the final two lines read "Validated table assumptions for ...".
Results: The postprocess script has different assertions about the results based on each software, e.g. checking that real behavior matches documented behavior or that different configurations behave the same. The output will list all these assertions, color-coded on whether they are as documented or whether they diverge.
Then the data is specifically checked against the behavior presented in Table 1. The corresponding code can be found in `evaluate/functionality/postprocess.py:394f` and `450f` in the `_check_table_assumptions...` functions.
Within each function, assertions are grouped by columns in the results table then rows. For example, the assertions in lines 396/397 and 454-460 check the results for the `SNI=I,H=I` case.
*Running this successfully explicitly confirms claim **(C1)**.*

A.4.3 Major Claims Artifact II

(C2): We claim that Table 2 presented in Section 4 is accurate for the versions listed in appendix A.

A.4.4 Experiments Artifact II

(E3): Verifying the Table *[10 human-minutes + 10 compute-minutes + 2GB disk]*
Preparation: Install the software according to appendix A.3.1.
Execution: Start the container. This will automatically run all tests against all web server configurations.

```
docker run --rm -v ./out:/code/out
 ↪ --privileged --name authdotsche
 ↪ authdotsche:latest
```

Alternatively, the provided file `run.sh` fully automates both steps, including the preparation.
Results: The experiment outputs its results both as console output and also as an HTML table. The resulting HTML table contains the full list of test cases and test results, which corresponds to the results presented in Table 2. The table will be saved in the mounted directory `./out/`.
You can verify the results by hand: Each column `A -> BC` describes the test case, where a ticket obtained with SNI and `Host` header A is then used with SNI B and `Host` C[1]. The first column `A->AA` (Ticket from Host A to SNI=A,Host=A) maps to the left most table column and so on.
Each row describes a different configuration: For example,

```
closedlitespeed_subdomains_certa_defaulta
    _strict_default
```

uses LiteSpeed, the two virtual hosts are on subdomains, the initial host[2] requires a certificate, is set as the default host and finally the software-specific `strict` setting is enabled.
The generated `results.html` includes a search field at the bottom, which filter the rows by regex, to improve usability. If you search `apache` you get all the `apache` variants, but you will not have to check all of them separately, as cells will be annotated with a colored tooltip, when behavior differs between variants (i.e. TLS 1.2 vs 1.3, A with client authentication vs freely accessible A, domain vs subdomain). For example, `apache A->nA` (so the SNI=none, Host=I column) gets marked because there is diverging behavior based on which host is set as a default, which matches the cross annotation in Table 2 at that entry.
*Verifying the table confirms claim **(C2)**.*

A.4.5 Major Claims Artifact III

(C3): We claim that our software scans pairs of virtual hosts with the same STEK, following the strategy outlined in Section 5.1.

(C4): We claim that, combined with human effort, this was used to retrieve the results presented in Section 5.2 and can be used to run the study again.

(C5): We claim that the suggested countermeasures in Section 7 are sensible.

A.4.6 Experiments Artifact III

(E4): Run local scan *[2 human-minutes + 5 compute-minutes]*:
Preparation: Run `./setup.sh` in background.
Execution: Run `./0_all_in_one.sh`
Results: The script will invoke our large-scale scanning pipeline.[3] It demonstrates all steps of the process, including initial resolution and ticket prefixes, importing a graph DB for querying scan candidates, performing the actual scan by using session tickets cross hosts and subsequent analysis. The relevant output is the resulting folder `analysisdump` (cf. **E2**). Note that the code also contains automated classification (`UNSAFE`), but all actual results stem from the manual part (`LOOK_AT_METRICS`).
*Running this successfully addresses claim **(C3)**.*

(E5): Examining the local results *[15 human-minutes]*:
Preparation: Run **(E4)**.
Execution: Manually examine the contents of the `analysisdump` folder.
Results: The folder contains a list of all potentially successful resumptions across hosts, grouped by ASes. Within, each path `./<target IP>/<source IP>/` contains a separate scan result. I.e., the folder `./172.19.0.5/172.19.0.3/` contains the different resulting HTML documents when resuming at `.5` with a ticket from `.3`: `0_initial.html` is the original page at `.3`, without a ticket. `1_resumed.html` contains the page received from `.5` after resumption. `2_*_supposed_origin.html` contains the closest HTML match, for `1_resumed.html`, based on different metrics. `_meta.md` summarizes these findings, including which domain we believe to have encountered.
We then manually examined these cases to find the vulnerable hosts. *Comprehending this workflow addresses claim **(C4)**.*

(E6): Retry with different countermeasures *[see above]*:
Preparation: Enable one of the countermeasures by editing `ae-dummy-servers/docker-compose.yml` in line 12f.
Execution: Re-run experiments **(E4)**,**(E5)**.

[1] n = not given, X = unavailable domain

[2] **I** in paper, **A** in code

[3] Changes have been made for running locally, but are marked with comments “AE Version”

Results: Running the previous experiments with different countermeasures enabled will yield different results (or none at all), indicating that fewer or no potentially unsafe resumptions were possible in the first place. *This supports claim **(C5).***

A.5 Notes on Reusability

Artifact III Our large-scale scanning pipeline is quite fragile at large sample sizes and we have experienced memory leaks, which may require occasional restarts. We also used Neo4J community version, leading to a workaround, as it only allows one database per container. Note, that the generated data may be upwards of 1TB, although our scripts have options for filtering out certain fields from zgrab etc. Nonetheless, the full pipeline can be easily restored from our submitted version to re-run the experiments if desired.

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/usenixsec2025/`.