
# USENIX Security '24 Artifact Appendix: SnailLoad: Exploiting Remote Network Latency Measurements without JavaScript

Stefan Gast, Roland Czerny, Jonas Juffinger, Fabian Rauscher, Simone Franza, Daniel Gruss
Graz University of Technology

## A Artifact Appendix

### A.1 Abstract

We present SnailLoad, a new side-channel attack where the victim loads an asset, e.g., a file or an image, from an attacker-controlled server, exploiting the victim's network latency as a side channel tied to activities on the victim system, e.g., watching videos or websites. SnailLoad requires no JavaScript, no form of code execution on the victim system, and no user interaction but only a constant exchange of network packets, e.g., a network connection in the background. SnailLoad measures the latency to the victim system and infers the network activity on the victim system from the TCP acknowledgment latency variations.

The artifact demonstrates the timing side-channel and shows how streaming YouTube videos or opening various websites causes characteristic patterns in the TCP acknowledgment latencies. We showcase an HTTP web server offering latency traces in a BMP file, generated on the fly while that BMP file is being downloaded.

### A.2 Description & Requirements

#### A.2.1 Security, privacy, and ethical concerns

Our artifact does not perform any destructive steps. It creates network latency traces correlating to activity on the internet connection, possibly also including activity of other users sharing the same connection or the same last mile bottleneck. While the traces do not directly contain sensitive information, privacy critical information can be derived from them, as described in the fingerprinting attacks in the paper. We therefore recommend to evaluate the artifact on an otherwise idle internet connection, with an exclusive last mile, like a FTTH connection.

#### A.2.2 How to access

The source code of the artifact is available at https://github.com/IAIK/SnailLoad, with the evaluated version of the artifact being available under https://github.com/IAIK/SnailLoad/tree/93612789e8a69d1340a4bf426fe3c42a6ebafb06. We also run the artifact at http://demo.snailload.com, using the source code provided at https://github.com/IAIK/SnailLoad.

#### A.2.3 Hardware dependencies

The artifact server can be accessed from any notebook or desktop computer on a home internet connection. The internet connection should be idle, with other devices being disconnected or disabled. For best results, evaluate on an ADSL or a slower (below 100 Mbit/s) FTTH connection.

Optionally, to run the artifact on an own server, a remote native or virtual server is required. The server must have reasonably stable ping times and must be reachable on a chosen TCP port from the client without a reverse proxy. We tested our server code on multiple hosters and on our university infrastructure.

#### A.2.4 Software dependencies

The client accessing the artifact server must have a graphical web browser installed. We tested Mozilla Firefox and Google Chrome.

To run the artifact on an own server, the artifact has to be compiled from source. For this, `gcc`, `make` and `libc-dev` are required on the machine used to set up the remote server. On the server, a Linux installation with SSH access and the ability to execute native code is required.

#### A.2.5 Benchmarks

None.

### A.3 Set-up

#### A.3.1 Installation

The following steps are only required for setting up the artifact on an own server.

1. Download the source code from https://github.com/IAIK/SnailLoad.
2. Change into the `SnailLoad/demo_server` directory: `cd SnailLoad/demo_server`.
3. Compile the server binary: `make`

4. Copy the `demo_server` binary to the server.
5. Log in to the server via SSH.
6. If the server binary should use a TCP port below 1024, adjust the capabilities: `sudo setcap cap_net_bind_service+ep demo_server`
7. Start the server binary on the designated TCP port: `./demo_server -p $PORT`, with `$PORT` being the port number.

#### A.3.2 Basic Test

1. Access the server in your browser, e.g., http://demo.snailload.com/plot.bmp or http://1.2.3.4:8080/plot.bmp.
2. A BMP image should load slowly, showing a vertical live trace of the current network latency. With an otherwise idle internet connection, the trace should be rather flat, with only occasional peaks.

### A.4 Evaluation workflow

#### A.4.1 Major Claims

**(C1):** A malicious TCP server can observe network activity on user's internet connection.
**(C2):** Streaming YouTube videos causes distinct latency patterns.
**(C3):** Opening websites causes distinct latency patterns.

#### A.4.2 Experiments

**(E1):** Basic activity detection [3 human-minutes]:
**How to:** Reload the BMP file in the browser. While the file is loading, open https://fast.com in another browser window.
**Results:** While https://fast.com is performing the speed test, latencies increase significantly in the trace.

**(E2):** Observing YouTube buffering [5 human-minutes]:
**How to:** Reload the BMP file in the browser. While the file is loading, open https://www.youtube.com/watch?v=1WEAJ-DFkHE&hd=1 in another browser window. Start the video and ensure it is playing with at least Full HD resolution.
**Results:** Latencies increase significantly whenever the video is buffering, *i.e.*, the gray progress bar in the YouTube player advances. Over time, this results in a characteristic pattern of low and high latencies in the trace.

**(E3):** Distinguishing websites [10 human-minutes]:
**How to:** Reload the BMP file in the browser. While the file is loading, open https://amazon.com in another browser window. After Amazon has loaded, open https://google.com. Repeat if desired.
**Results:** Loading Amazon leads to an extended period with higher latencies, whereas loading Google only results in minor latency spikes.

### 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/usenixsec2024/.