USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Lancet: A Formalization Framework for Crash and Exploit Pathology

Qinrun Dai, Kirby Linvill, Yueqi Chen, and Gowtham Kaki,
University of Colorado Boulder

https://www.usenix.org/conference/usenixsecurity25/presentation/dai

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

USENIX Security '25 Artifact Appendix: Lancet: A Formalization Framework for Crash and Exploit Pathology

A Artifact Appendix

We aim to apply for Available, Functional and Reproduced Badges.

A.1 Abstract

This artifact is applying for an Artifacts Available Badge, an Artifacts Functional Badge, and a Results Reproduced Badge. For the Available badge, the source code and scripts can be found at the Zenodo [10.5281/zenodo.15611790]. To make it easier for evaluators to reproduce the experiments, we have provided Docker containers via GitHub Packages with public access. Ongoing updates and further development will be released through the same GitHub repository. For the Functional and Reproduced badges, we will use CVE-2019-6977, House of Einherjar, CVE-2024-41965, CPV15, cases in Juliet Test Suite, and CVE-2023-33476, to demonstrate workflows of root cause diagnosing and exploit primitives identification. All experiments are expected to be conducted in Docker container.

A.2 Description & Requirements

In this section, we first describe whether reproducing our artifacts will risk the evaluator's machine security, followed by approaches to accessing our artifacts. Then, we describe hardware dependencies and software dependencies before listing the benchmarks.

A.2.1 Security, privacy, and ethical concerns

`Lancet` analyzes buggy or exploitable softwares' behavior using Intel Pintools. It instruments programs at runtime without interfering their original behavior. To address any concerns about executing code, we provide Docker images that do not require privileged flags. Evaluators can pull these images from GitHub Container Registry and reproduce our results in an isolated environment, ensuring the host machine's safety and privacy (see Section A.2.1). Moreover, all exploits and bug instances have been modified to be harmless and cannot escape from the Docker container. As a result, there are no security, privacy, or ethical risks for the community.

A.2.2 How to access

The artifact can be accessed at `https://zenodo.org/records/15611790`.

A.2.3 Hardware dependencies

To reproduce `Lancet` and obtain reasonably precise results, we recommend the following minimum hardware configuration: (1) an Intel CPU of similar performance to the i5-13600KF, (2) at least 16GB of RAM, and (3) a minimum of 50GB of disk space with a Docker-enabled environment.

A.2.4 Software dependencies

We recommend conducting the evaluation on the Ubuntu Linux distribution, preferably version 22.04 Desktop with Docker installed, which matches the development environment used for `Lancet`. The specific libc version associated with the distribution affects the precise memory layout of the bugs used in our artifact evaluation. While we provide a Docker image to eliminate environmental discrepancies and improve reproducibility, we conservatively advise performing all reproduction efforts within the suggested environment to ensure maximum consistency and accuracy.

A.2.5 Benchmarks

The exploits and bug instances used as test cases are embedded within the provided Docker images and are also publicly accessible followig the references provided in the paper. To evaluate runtime overhead, we use two modes: a baseline mode (without our reasoning logic) and a full mode. This comparison allows us to isolate the overhead introduced by our approach from that of the underlying Pin-based instrumentation. The difference in execution time between these two modes represents the overhead reported in the paper.

A.3 Set-up

In this section, we focus on how to test `Lancet` using the Docker images we provided.

A.3.1 Installation

Install Docker using your system's package manager (*e.g.*, apt or yum), then pull all images from the GitHub repository: https://github.com/a85tract/Lancet.

A.3.2 Basic Test

N/A

A.4 Evaluation workflow

Evaluate whether `Lancet` identifies the root cause with the the correct stdout output. More specific details, including root cause references for each bug and exploit, can be found in the GitHub repository.

A.4.1 Major Claims

(C1): `Lancet` can accurately analyze the given bugs and exploits.

(C2): The time consumption of each analysis task is in line with what is claimed in Table 1, Table 2 and Table 5 of the paper.

A.4.2 Experiments

(E1): [Bug CPV15] [10 human-minutes + 10 compute-minutes + 5GB disk]:
Preparation: `docker pull ghcr.io/a85tract/lancet:cpv15`
Execution: (1) `docker run ghcr.io/a85tract/lancet:cpv15 baseline` (2) `docker run ghcr.io/a85tract/lancet:cpv15 lancet`
Results: (1) Each run reports total time consumption; the difference between the two indicates the time taken by `Lancet`. (2) `Lancet` should report: ① allocation's stacktrace of corrupted memory buffer ② cell owner id and value owner id of target memory cell ③ pointer untrusted dereference

(E2): [Bug CVE-2024-41965] [10 human-minutes + 10 compute-minutes + 5GB disk]
Preparation: `docker pull ghcr.io/a85tract/lancet:cve_2024_41965`
Execution: (1) `docker run ghcr.io/a85tract/lancet:cve_2024_41965 baseline` (2) `docker run ghcr.io/a85tract/lancet:cve_2024_41965 lancet`
Results: (1) Each run reports total time consumption; the difference between the two indicates the time taken by `Lancet`. (2) `Lancet` should detect double free and log the exact instruction numbers executed prior to the double free bug.

(E3): [Bug CVE-2019-6977] [10 human-minutes + 10 compute-minutes + 5GB disk]
Preparation: `docker pull ghcr.io/a85tract/lancet:cve_2019_6977`
Execution: (1) `docker run ghcr.io/a85tract/lancet:cve_2019_6977 baseline` (2) `docker run ghcr.io/a85tract/lancet:cve_2019_6977 lancet`
Results: (1) Each run reports total time consumption; the difference between the two indicates the time taken by `Lancet`. (2) `Lancet` should: ① detect out-of-bounds writes by logging ADD instructions that violate memory safety rules within paper's reasoning model ② allocation's stacktrace of corrupted memory buffer

(E4): [Exploit CVE-2023-33476] [15 human-minutes + 10 compute-minutes + 5GB disk]
Preparation: `docker pull ghcr.io/a85tract/lancet:cve_2023_33476`
Execution: (1) In order to reproduce this experiment more smoothly, we recommend disable ASLR on the host (outside docker container) by: `echo 0 > /proc/sys/kernel/randomize_va_space` (need root permission). This operation is reversible by echo 2 to the same file. (2) Launch terminal A, run: `docker run -it ghcr.io/a85tract/lancet:cve_2023_33476`, then run: `cd /app/src/minidlna-git` inside the container (3) Launch another terminal B, use docker exec to enter the same contianer, then run: `cd /app/exploits` (4) In terminal A, run: `/app/pin/pin -t /app/lancet.so -- ./minidlnad -R -f minidlna.conf -d`, and wait until no new logs are printed (about 1 minute) (5) run: `python3 tpoison-nopie-x64_reverse-shell.py 127.0.0.1 --system_addr 0x7ffff63a52f0 --got_addr 0x45e150`, in terminal B
Optional: We have also prepared an ASan version; detailed steps can be found in the GitHub README.
Results: `Lancet` should report: ① memmove violation ② logs such as: "[exploit primitive] allocation in .got.plt region", indicating memory overlapping ③ logs such as: "[exploit primitive] write in .got.plt region" indicates expolit is trying to overwrite GOT table

(E5): [Juliet Test Suite & How2heap] [30 human-minutes + 10 compute-minutes + 5GB disk]
Preparation: `docker pull ghcr.io/a85tract/lancet:juliet_how2heap`
Execution: (1) `docker run ghcr.io/a85tract/lancet:cve_2019_6977 <tool> <target> <case>` tool options: `Asan`, `lancet`, `baseline`, target options: `how2heap`, `juliet`, how2heap case options: `fastbin_reverse_into_tcache`, `house_of_einherjar`, `poison_null_byte`, juliet case options: `CWE121_Stack_Based_Buffer_Overflow__CWE129_fgets_01-bad`, `CWE416_Use_After_Free__new_delete_class_01-bad`, ... (2) Hacknote: run `docker run -it --entrypoint /bin/bash ghcr.io/a85tract/lancet:juliet_how2heap`, then run commands inside the docker `cd hacknote; python3`

`exp.py`. Wait until no more output is generated and receive `Your choice :$`, then input 4 with enter.
Results: (1) Each run reports total time consumption; the difference between baseline and lancet indicates the time taken by `Lancet`. (2) Due to space constraints, we'll place the remaining content in the GitHub repo's README file.

(E6): [Bug OSV-2024-204] [10 human-minutes + 10 compute-minutes + 5GB disk]
Preparation: `docker pull ghcr.io/a85tract/lancet:osv_2024_204`
Execution: (1) `docker run ghcr.io/a85tract/lancet:osv_2024_204 baseline` (2) `docker run ghcr.io/a85tract/lancet:osv_2024_204 lancet`
Results: (1) Each run reports total time consumption; the difference between the two indicates the time taken by `Lancet`. (2) `Lancet` should detect DANGLINGPTR and log the exact instruction numbers executed prior to the use of freed pointer.

(E7): [Bug FFmpeg, #11228] [10 human-minutes + 10 compute-minutes + 5GB disk]
Preparation: `docker pull ghcr.io/a85tract/lancet:ffmpeg_11228`
Execution: (1) `docker run ghcr.io/a85tract/lancet:ffmpeg_11228 baseline` (2) `docker run ghcr.io/a85tract/lancet:ffmpeg_11228 lancet`
Results: (1) Each run reports total time consumption; the difference between the two indicates the time taken by `Lancet`. (2) `Lancet` should detect NLLPTRDEREF/UNTRUSTEDPTRDEREF.

(E8): [Bug FFmpeg, #10749] [10 human-minutes + 10 compute-minutes + 5GB disk]
Preparation: `docker pull ghcr.io/a85tract/lancet:ffmpeg_10749`
Execution: (1) `docker run ghcr.io/a85tract/lancet:ffmpeg_10749 baseline` (2) `docker run ghcr.io/a85tract/lancet:ffmpeg_10749 lancet`
Results: (1) Each run reports total time consumption; the difference between the two indicates the time taken by `Lancet`. (2) `Lancet` should detect NLLPTRDEREF .

(E9): [Bug OSV-2023-1276] [10 human-minutes + 10 compute-minutes + 5GB disk]
Preparation: `docker pull ghcr.io/a85tract/lancet:osv_2023_1276`
Execution: (1) `docker run ghcr.io/a85tract/lancet:osv_2023_1276 baseline` (2) `docker run ghcr.io/a85tract/lancet:osv_2023_1276 lancet`
Results: (1) Each run reports total time consumption; the difference between the two indicates the time taken by `Lancet`. (2) `Lancet` should detect DANGLINGPTR and log the exact instruction numbers executed prior to the use of freed pointer.

(E10): [Bug CVE-2024-43374] [10 human-minutes + 10 compute-minutes + 5GB disk]
Preparation: `docker pull ghcr.io/a85tract/lancet:cve_2024_43374`
Execution: (1) `docker run ghcr.io/a85tract/lancet:cve_2024_43374 baseline` (2) `docker run ghcr.io/a85tract/lancet:cve_2024_43374 lancet`
Results: (1) Each run reports total time consumption; the difference between the two indicates the time taken by `Lancet`. (2) `Lancet` should detect DANGLINGPTR and log the exact instruction numbers executed prior to the use of freed pointer.

(E11): [Bug GPAC, #2701] [10 human-minutes + 10 compute-minutes + 5GB disk]
Preparation: `docker pull ghcr.io/a85tract/lancet:gpac_2701`
Execution: (1) `docker run ghcr.io/a85tract/lancet:gpac_2701 baseline` (2) `docker run ghcr.io/a85tract/lancet:gpac_2701 lancet`
Results: (1) Each run reports total time consumption; the difference between the two indicates the time taken by `Lancet`. (2) `Lancet` should detect DANGLINGPTR and log the exact instruction numbers executed prior to the use of freed pointer.

(E12): [Bug GPAC, #2583] [10 human-minutes + 10 compute-minutes + 5GB disk]
Preparation: `docker pull ghcr.io/a85tract/lancet:gpac_2583`
Execution: (1) `docker run ghcr.io/a85tract/lancet:gpac_2583 baseline` (2) `docker run ghcr.io/a85tract/lancet:gpac_2583 lancet`
Results: (1) Each run reports total time consumption; the difference between the two indicates the time taken by `Lancet`. (2) `Lancet` should detect DANGLINGPTR and log the exact instruction numbers executed prior to the use of freed pointer.

(E13): [Bug PHP, #16595] [10 human-minutes + 10 compute-minutes + 5GB disk]
Preparation: `docker pull ghcr.io/a85tract/lancet:php_16595`
Execution: (1) `docker run ghcr.io/a85tract/lancet:php_16595 baseline` (2) `docker run ghcr.io/a85tract/lancet:php_16595 lancet`
Results: (1) Each run reports total time consumption; the difference between the two indicates the time taken by `Lancet`. (2) `Lancet` should detect DANGLINGPTR and log the exact instruction numbers executed prior to the use of freed pointer.

(E14): [Bug OSV-2024-96] [10 human-minutes + 10 compute-minutes + 5GB disk]
Preparation: `docker pull ghcr.io/a85tract/lancet:osv_2024_96`
Execution: (1) `docker run ghcr.io/a85tract/lancet:osv_2024_96 baseline` (2)

`docker run ghcr.io/a85tract/lancet:osv_2024_96 lancet`

Results: (1) Each run reports total time consumption; the difference between the two indicates the time taken by `Lancet`. (2) `Lancet` should detect CROSSBOUNDARY.

(E15): [Bug PHP, #76041] [10 human-minutes + 10 compute-minutes + 5GB disk]

Preparation: `docker pull ghcr.io/a85tract/lancet:php_76041`

Execution: (1) `docker run ghcr.io/a85tract/lancet:php_76041 baseline` (2) `docker run ghcr.io/a85tract/lancet:php_76041 lancet`

Results: (1) Each run reports total time consumption; the difference between the two indicates the time taken by `Lancet`. (2) `Lancet` should detect NLLPTRDEREF.

(E16): [Bug CVE-2004-1287] [10 human-minutes + 10 compute-minutes + 5GB disk]

Preparation: `docker pull ghcr.io/a85tract/lancet:cve_2004_1287`

Execution: (1) `docker run ghcr.io/a85tract/lancet:cve_2004_1287 baseline` (2) `docker run ghcr.io/a85tract/lancet:cve_2004_1287 lancet`

Results: (1) Each run reports total time consumption; the difference between the two indicates the time taken by `Lancet`. (2) `Lancet` should detect CROSSBOUNDARY.

(E17): [Bug CVE-2007-1001] [10 human-minutes + 10 compute-minutes + 5GB disk]

Preparation: `docker pull ghcr.io/a85tract/lancet:cve_2007_1001`

Execution: (1) `docker run ghcr.io/a85tract/lancet:cve_2007_1001 baseline` (2) `docker run ghcr.io/a85tract/lancet:cve_2007_1001 lancet`

Results: (1) Each run reports total time consumption; the difference between the two indicates the time taken by `Lancet`. (2) `Lancet` should detect CROSSBOUNDARY.

(E18): [Bug CVE-2012-2386] [10 human-minutes + 10 compute-minutes + 5GB disk]

Preparation: `docker pull ghcr.io/a85tract/lancet:cve_2012_2386`

Execution: (1) `docker run ghcr.io/a85tract/lancet:cve_2012_2386 baseline` (2) `docker run ghcr.io/a85tract/lancet:cve_2012_2386 lancet`

Results: (1) Each run reports total time consumption; the difference between the two indicates the time taken by `Lancet`. (2) `Lancet` should detect CROSSBOUNDARY.

A.5 Correction

During the artifact evaluation, we identified two typographical errors in Table 1 of the paper. The FCS report for the PHP #76041 should be `NLLPTRDEREF`, and the report for the FFmpeg, #10749 should also be `NLLPTRDEREF`.

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/.