USENIX Security '24 Artifact Appendix: SSRF vs. Developers: A Study of SSRF-Defenses in PHP Applications

Malte Wessels*†, Simon Koch*†, Giancarlo Pellegrino‡, Martin Johns†
† Technische Universität Braunschweig
‡ CISPA Helmholtz Center for Information Security
{malte.wessels, simon.koch, m.johns}@tu-braunschweig.de, pellegrino@cispa.de

A Artifact Appendix

A.1 Abstract

A new PHP code property graph (CPG) generator. It is based on PHP bytecode lifted directly from the PHP interpreter. Additionally, we supply a static analysis pipeline (slicing and string reconstruction) to find SSRF candidates. We are applying for the functional badge.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Security and Ethics Concerns: We use a publicly available PHP web shell with an SSRF candidate flow to demonstrate the functionality. Since a web shell is vulnerable by design, it should not be hosted. *Privacy:* We pull dependencies from GitHub, Docker, and via sbt. This requires connections to these servers.

A.2.2 How to access

An overview is available at: https://github.com/SSRF-vs-Developers.

Version-fixed docker files for our code property graph generation and experiment runner are available at: https://github.com/SSRF-vs-Developers/CpgGeneration/tree/artifact.

SURFER, an SSRF candidate detection tool to run on the generated PHP CPG, is available at: https://github.com/SSRF-vs-Developers/surfer/tree/artifact.

The GitHub organization contains an overview, which we used as an entry point for this artifact evaluation[1].

A.2.3 Hardware dependencies

None, for the sake of demonstration we use small examples.

A.2.4 Software dependencies

A recent Linux system with docker and git installed.

A.2.5 Benchmarks

None.

A.3 Set-Up

We provide several tools which are automatically built by the docker containers.

PHP-src We patched the PHP interpreter to output more data in its bytecode debug output.

CPG A code property graph generator based on PHP bytecode.

Slicer A slicing utility for this CPG.

Plotter A plotting utility for slices.

SURFER Our tool to identify SSRF candidates in these CPGs.

Optional: Install Joern

Install Joern[2] via its README or docker. We can confirm that Joern version 2.0.223 is suitable, but we recommend trying the latest version first. Pulling the docker image might require a GitHub login[3].

```
# pull docker image
docker pull ghcr.io/joernio/joern:nightly
```

*Both authors contributed equally to this research.

[1] https://github.com/SSRF-vs-Developers/.github/tree/ed093a0443fefd4a8a2d8c134df813e80a6dfa5a/profile

[2] https://github.com/joernio/joern/

[3] https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-container-registry

A.3.1 Installation

Our toolchains are docker-based. We will provide recent toolchain versions in the GitHub organization at https://github.com/PHP-CPG. However, for the purpose of this artifact evaluation, we created dockerfiles that pin the versions. To build these, clone their repository: https://github.com/SSRF-vs-Developers/CpgGeneration. We provide scripts that create (build and test) these docker containers. Change the directory to their folders and run them in this order:

1. CPG/resources/docker/PHP-StringPatched/create.sh
2. CPG/resources/docker/multilayer-php-cpg/create.sh
3. ExperimentRunner/template/create.sh

Each dockerfile will build the toolchain, including pulling dependencies and **running test cases**. We, finally, build our tool SURFER by running the following commands:

- Clone https://github.com/SSRF-vs-Developers/surfer
- `cd resources/docker`
- `./create.sh`

A.3.2 Basic Test

The repositories ships with the most basic SSRF example to test the pipeline. Run this command: `./surfer_docker_run.sh ./basictestfiles/in/ ./basictestfiles/cpg ./basictestfiles/out`

This creates a cpg in the *cpg* folder. Additionally, a *testproject.json* file is created in the *out* folder. This JSON contains one candidate:

```
"reversedString": ["<G:_GET>[x]"],
"sink": ["/in/testproject/test.php", "1"],
"sinkName": "file_get_contents",
"sources": [["/in/testproject/test.php", "1"]]
```

Additionally, a dot file is generated, which visualizes our slice. It can be rendered with utilities like Graphviz.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *We provide a bytecode PHP code property graph (CPG) generator.*
(C2): SURFER *can find SSRF candidates in these CPGs.*

A.4.2 Experiments

We will use one of the repositories from our dataset to show the functionality. For the sake of demonstration, instead of using a productive application as an analysis subject, we use a publicly available PHP reverse shell, which we found in our dataset and classified as a 'hacking tool'[4].

E1 addresses C1, and E2 addresses C2, respectively.

(E1): *[$\sim$ 1 compute-minute + $<$ 100 MB disk]*: Creating a CPG from a project: We fetch the source code and create a CPG. Additionally, this runs SURFER on the created CPG.
Preparation: Install docker, wget, and a tool to unpack zip files, e.g., unzip.
How to: First, cd into the 'in' folder: 'surfer/basictestfiles/in'. Then download and extract https://github.com/ivan-sincek/php-reverse-shell/archive/refs/tags/v2.6.zip.
Execution: Rerun the command from A.3.2: `./surfer_docker_run.sh ./basictestfiles/in/ ./basictestfiles/cpg ./basictestfiles/out`
Results: Cd into the 'basictestfiles/cpg' folder and observe that a .cpg file was created. Optionally: Load it into joern[5] via 'joern file.cpg' and run queries, e.g., 'cpg.call.size'. If using dockererized joern:

```
docker run --rm -it -v /tmp:/tmp -v
    $(pwd):/app:rw -w /app -t ghcr.io
    /joernio/joern:nightly joern /app
    /file.cpg
joern> cpg.call.size
```

(E2): *[*SURFER*] [1 min]*: The previous experiment also ran SURFER automatically. Navigate to the 'out' folder and confirm that a JSON file with the project's name was created. It should have 2 SSRF candidate flows under the 'candidates' key. Each candidate contains a reversed string. Additionally, .dot files are created to visualize the slice through our CPG.

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/.

[4] https://github.com/ivan-sincek/php-reverse-shell/
[5] https://github.com/joernio/joern/