USENIX Security '24 Artifact Appendix: Argus: All your (PHP) Injection-sinks are belong to us.

Rasoul Jahanshahi
Boston University
rasoulj@bu.edu

Manuel Egele
Boston University
megele@bu.edu

A Artifact Appendix

A.1 Abstract

Our artifact facilitates building and running Argus on the PHP interpreter. We packaged the artifact in a set of Docker containers. There are no restrictions on the CPU architecture or operating system for building and running Docker containers.

In this appendix, we describe the workflow of analyzing the PHP interpreter v5.6, identifying potential PHP APIs vulnerable to insecure deserialization, and validating the identified APIs. Finally, we demonstrate that extending a static analysis tool (i.e., Psalm) leads to identifying a previously unknown security vulnerability.

A.2 Description & Requirements

This section describes the requirements for running Argus' artifact.

A.2.1 How to access

Download the artifact from: `https://zenodo.org/records/12811284`

A.2.2 Hardware dependencies

None

A.2.3 Software dependencies

Docker

A.2.4 Benchmarks

In the artifact of Argus' evaluation, we use PHP version 5.6 and Psalm v4 as the benchmark.

A.2.5 Security, privacy, and ethical concerns

This artifact only operates on local files inside the container and does not require any communication with the Internet. One can run the container with the flag `-network none` option to isolate the container.

A.3 Set-up

A.3.1 Installation

Our instructions are based on Docker containers. Please use this link to install Docker and run these containers.

A.3.2 Prepration

In order to run our artifact, please first download the artifact from the link provided in the github repository. Next, please run the following commands in a terminal:

```
$ docker import argus-artifact.tar
argus-artifact:1.0
```

For the second artifact, which includes the extended Psalm, please unzip `phase-3` and run the following command in the repository's directory:

```
$ cd phase-3 && ./run.sh build
```

The above command will build the container for the last phase of the artifact evaluation (i.e., extending Psalm).

A.3.3 Basic Test

The basic test for this evaluation includes running the containers as well as the preparation script inside the containers. In order to run the artifacts, please execute the following commands in the terminal:

```
$ docker run --rm --workdir /home -it
argus-artifact:1.0 bash
```

Finally, run the following commands inside the container to perform a basic test of Argus:

```
$ cd step-1 && ./prepare.sh
```

The above process may take a few minutes to complete. At the end, you should see the message, *"The preparation is done."*.

A.4 Evaluation workflow

This section describes the workflow for evaluating the artifact for Argus.

A.4.1 Major Claims

Argus is a static-dynamic analysis of the PHP interpreter to identify vulnerable PHP APIs to injection vulnerabilities. According to our paper, we prove the following claims regarding our artifact's evaluation and its results:

(C1): *Argus identifies the PHP APIs vulnerable to injection vulnerabilities, such as insecure deserialization, by analyzing the PHP interpreter. The analysis in experiment (E1), as detailed in Section A.4.2.1 and Table 5 of our paper, substantiates this claim.*

(C2): *Argus improves the existing security analysis tools, such as Psalm, by extending the list of taint sinks. We prove this claim in experiment (E2), as detailed in Table 4 of our paper.*

A.4.2 Experiments

(E1): *[Identification] [15 human-minutes + 30 compute-minutes + 5GB disk]:* In this experiment, Argus statically analyzes and generates the call-graph for the PHP interpreter v5.6. Next, Argus merges the statically generated call-graph with the previously recorded dynamic traces of running PHP unit tests. Finally, Argus runs the reachability analysis to detect PHP APIs vulnerable to insecure deserialization as well as perform the validation.

How to: First, run the `argus-artifact` container by executing the following command:

```
$ docker run --rm --workdir /home -it
argus-artifact:1.0 bash
```

Preparation: As mentioned in Section A.3.3, you can run the preparation script inside the container by executing the following command:

```
$ cd step-1 && ./prepare.sh
```

Execution: In order to run the analysis of Argus, please execute the following command inside the container:

```
$ cd step-1 && ./run.sh
```

The above command will analyze the PHP interpreter and identify the APIs that are vulnerable to insecure deserialization. This script will write the set of potential APIs to a file named `step-2/list`.

In the next step, Argus validates the set of APIs detected in the previous step. To initiate the validation process, execute the following command:

```
$ cd step-2 $$ ./run.sh
```

The above command will execute the validation process for the PHP APIs in the `list` file. Argus generates a PHP snippet for each API, as described in Listing 4 of our paper, and validates the API by executing the snippet and analyzing its execution.

(E2): *[Extension] [15 human-minutes + 15 compute-minute + 1GB disk]:* In this experiment, we extend Psalm with the results of Argus and demonstrate that extended Pslam can identify previously unknown vulnerabilities.

How to: First, run the `extended-psalm` container by executing the following command:

```
$ cd phase-3 && ./run shell
```

Execution: Inside the container, please run the following command to run Psalm's anlaysis without any extension:

```
$ ./run.sh
```

The command above prints out the set of detected vulnerabilities in the ImageMagick plugin for WordPress. Since Psalm is not extended yet, it cannot identify the insecure deserialization vulnerability stated in Table 4 of our paper (i.e., CVE-2022-2441). To extend Psalm and run Psalm+Argus analysis, please run the following command:

```
$ ./run.sh 1
```

The command above will automatically extend Psalm to include `is_executable` as a sink for insecure deserialization and run the analysis. After finishing the analysis, Psalm+Argus prints out two newly identified insecure deserialization vulnerabilities in ImageMagick as well.

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/.