USENIX'23 Artifact Appendix: Minimalist: Semi-automated Debloating of PHP Web Applications through Static Analysis

Rasoul Jahanshahi
Boston University
rasoulj@bu.edu

Babak Amin Azad
Stony Brook University
baminazad@cs.stonybrook.edu

Nick Nikiforakis
Stony Brook University
nick@cs.stonybrook.edu

Manuel Egele
Boston University
megele@bu.edu

A Artifact Appendix

A.1 Abstract

Our artifact facilitates building and running Minimalist for phpMyAdmin 4.0.0. We packaged the artifact in a set of Docker containers. There is no restriction on the CPU architecture or the operating system to build and run the Docker containers.

In this appendix, we describe the workflow of analyzing a PHP application (e.g., phpMyAdmin) using Minimalist, identifying the set of unnecessary functions according to prior user interaction, and debloating the PHP applications. Finally, we demonstrate that debloating PHP applications leads to reducing the size of the application as well as removing security vulnerabilities.

A.2 Description & Requirements

A.2.1 How to access

Download the Artifacts from: https://github.com/BUseclab/Minimalist/releases/tag/v1.0.1

A.2.2 Software dependencies

Docker and Docker compose

A.2.3 Benchmarks

In our artifact evaluation of Minimalist, we use phpMyAdmin v4.0.0 as the benchmark for our artifact evaluation.

A.3 Set-up

A.3.1 Installation

Our instructions are based around Docker containers. Please install Docker and Docker compose to run these containers:

- Docker https://docs.docker.com/get-docker/
- Docker-compose https://docs.docker.com/compose/install/

A.3.2 Prepration

In order to run our artifact, you need to download the required packages for Minimalist as well as Less is More artifact. To do so, please run the following command to download the required packages.

```
$ cat prepare.sh  # Examine the script

$ ./prepare.sh    # Run the prepare script
```

A.3.3 Basic Test

Our basic test involves building and running Minimalist on a sample PHP web application inside a Docker container. In order to run the basic test, please run the following command in the main directory of the artifact.

```
$ cat init.sh # Examine the init script...

$ ./init.sh   # Build and run the initial test
```

In case of a successful initial test, you should see the following message.

```
Basic Test was successful.
```

A.4 Evaluation workflow

A.4.1 Major Claims

Minimalist is a debloating mechanism for PHP web applications. According to our paper, we prove the following claims regarding the evaluation of our artifact and its results:

(C1): *Minimalist reduces the size of a given PHP web application (e.g., phpMyAdmin) in terms of lines of code. This claim is proven by the reduction in size of phpMyAdmin*

in experiment (E1), which is described in Section 4.4.1 as well as Figure 7 of our paper.

(C2): *Minimalist removes the security vulnerabilities in PHP applications by removing unnecessary features. This claim is proven in experiment (E2), which is described in Table 1 of our paper.*

A.4.2 Experiments

(E1): *[Debloat] [20 human-minutes + 1 compute-hour + 5GB disk]: In this experiment, Minimalist statically analyzes phpMyAdmin 4.0.0 and generates the call-graph. Next, Minimalist debloats the web application using Less is More interface.*

Preparation: None

Execution: The first step is running the Minimalist analysis on phpMyAdmin 4.0.0. To do so, run the following commands to download phpMyAdmin, create a Docker container, prepare the Docker environment, and run the analysis.

```
$ cat step_1.sh  # Examine step_1 script

$ ./step_1.sh    # Run step_1 script
```

At the end of this step, Minimalist generates the call-graph for phpMyAdmin 4.0.0. You can compare the results regarding the number of different function calls shown in the terminal with the numbers in the first three columns of Table 1 for phpMyAdmin 4.0.0.

The next step includes debloating phpMyAdmin using the Less is More (LIM) container. To do so, run the following commands to run the LIM container.

```
$ cat step_2.sh  # Examine step_2 script

$ ./step_2.sh    # Run step_2 script
```

After running the LIM container, you can import Minimalist results to LIM either manually or automatically. You can follow our tutorial in our Github repository to import the results manually. Run the following command to import the results automatically. Note that this process takes up to 20 minutes.

```
$ cat auto_import.sh #Examine import script

$ ./auto_import.sh   #Run import script
```

Before debloating, run the following command to measure the lines of code (LoC) for phpMyAdmin 4.0.0.

```
$ ./phploc.sh   # Run the phploc Docker
```

Furthermore, run the following command to perform a SQLi attack on phpMyAdmin 4.0.0. For a successful attack, the server takes more than five seconds to respond.

```
$ ./exploit.sh   # Run exploit script
```

In the last step, visit the following link to start the debloating process.
http://localhost:8086/admin/software_file/description
In the following link, click on the `add` button in the top right corner, fill out the form using the following inputs, and click `populate database`.

```
Software: phpMyAdmin
Version: 4.0.0
Web App Directory: /var/www/html/4.0.0/
Description: Artifact
```

After finishing the above process, click on the `Debloat functions` to start the debloating process. This process takes up to five minutes to complete. For more information, you can follow the visual tutorial in our Github repository.

After finishing the debloating process for phpMyAdmin 4.0.0, you can run the `phploc` script again to calculate the LoC for the debloated web application. As shown in Figure 7 in our paper, you can observe the reduction in LoC for phpMyAdmin 4.0.0 before and after debloating.

(E2): *[Attack] [10 human-minutes + 10 compute-minutes]: In this experiment, you perform a SQLi attack to examine the removed vulnerability from Minimalist-debloated phpMyAdmin.*

Prepration: To perform this experiment, do not stop the LIM container.

Execution: In order to perform the SQLi attack, run the following command in a separate terminal.

```
$ cat exploit.sh   # Examine exploit script

$ ./exploit.sh    # Run exploit script
```

Results: In the case of a failed attack, the response from phpMyAdmin is immediate. Otherwise, the server takes more than five seconds to respond.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.