A Artifact Appendix

A.1 Abstract

FUGIO is the first automatic exploit generation (AEG) tool for PHP object injection (POI) vulnerabilities. The artifact provides Docker images to reproduce the experiments performed in the paper. We tested these Docker images and scripts on a Ubuntu 18.04 machine. Each Docker container requires less than 5 GB of disk, but it requires more disk spaces for running FUGIO that stores identified POP chains and generates exploit objects to be fed to the fuzzer. We expect the artifact reproduces evaluations in Sections 7.2 and 7.3, producing Tables 1, 2, and 3 in the paper. Unfortunately, it might be hard to expect the same experimental results as the paper since FUGIO conducts fuzzing campaigns, and evaluations would be conducted in machines with different specifications.

A.2 Artifact check-list (meta-information)

- **Program:** We evaluated FUGIO on 30 PHP applications:
  - PHP 5.4: Contao CMS 3.2.4, Piwik 0.4.5, GLPI 0.83.9, Joomla 3.0.2, CubeCart 5.2.0, CMS Made Simple 1.11.9, Open Web Analytics 1.5.6, Vanilla Forums 2.0.18.5, SwiftMailer 5.0.1, SwiftMailer 5.1.0, Smarty 3.1.28, ZendFramework 1.12.20
  - PHP 5.6: PHPExcel 1.8.1, PHPExcel 1.8.2, Dompdf 0.8.0, Guzzle 6.0.0, WooCommerce 2.6.0, WooCommerce 3.4.0, Emailsubscribers 4.4.0, EverestForms 1.6.6 (w/ WordPress 5.0)
  - PHP 7.2: TCPDF 6.3.2, Drupal 7.78, SwiftMailer 5.4.12, SwiftMailer 6.0.0, Monolog 1.7.0, Monolog 1.18.0, Monolog 2.0.0, Laminas 2.11.2, Yii 1.1.20, TYPO3 9.3.0

  All benchmarks are included in the `benchmarks` directory. The artifact provides not only applications' source code also dump files of each application and its database for convenient settings.
- **Compilation:** FUGIO requires some libraries to be compiled. It needs only public compilers and the artifact provides all scripts to compile the libraries.
- **Run-time environment:** The artifact runs on Docker containers. We tested our Docker images and scripts on a Ubuntu 18.04 host machine. Given Docker images might work on any OS if it supports Docker.
- **Output:** After analyzing the target application, FUGIO generates 1) a PUT. When FUGIO identifies 2) a POP chain, it saves it as a file. If a POP chain reaches the sensitive sink, FUGIO reports the POP chain as 3) a probably exploitable chain. If the POP chain invokes the sink with a parameter containing the attack payload, FUGIO reports the POP chain as 4) an exploitable chain. All outputs are generated in the `Files/fuzzing/[APP_PATH.TIME]/PUT` directory.
  1. PUT: `put-head.php` and `put-body.php` are PUT files; `inst_PUT.php` is an instrumented PUT file for fuzzing the target application.
  2. POP chains: identified POP chains are stored as filename `procX_X_X_X_X_X.chain`.
  3. Probably exploitable chains: probably exploitable payloads are stored in the `PROBABLY_EXPLOITABLE` directory.
  4. Exploitable chains: exploitable payloads are stored in the `EXPLOITABLE` directory.

  During FUGIO identifies POP chains and generates their exploits, it periodically shows the progress to the console: how long FUGIO is running, how many POP chains are identified, how many POP chains are fed to the Fuzzer, how many probably exploitable payloads are generated, and how many exploitable payloads are generated.
- **Experiments:** We expect the artifact reproduces Tables 1, 2, and 3 in Sections 7.2 and 7.3. The artifact provides the `config.py` script for preparing the corresponding environment in which each experiment was conducted. However, it might be hard to expect the same experimental results since 1) FUGIO conducts fuzzing campaigns, which randomly produces results, and 2) evaluations would be conducted in machines with different specifications; our evaluations were performed on a Linux workstation equipped with 88 cores of CPUs and 384 GB of RAM.
- **How much disk space required (approximately)?:** Each Docker container does not require more than 5 GB of disk. However, FUGIO sometimes requires hundreds of GB depending on the target application since FUGIO identifies millions of POP chains.
- **How much time is needed to prepare workflow (approximately)?:** Preparing Docker containers and FUGIO takes less than an hour. Most of the time is spent on building the Docker image and installing dependencies.
- **How much time is needed to complete experiments (approximately)?:** The running time of FUGIO depends on the target application and the specification of the machine. For each application, Table 1 provides the time spent in running FUGIO on a machine equipped with 88 cores of CPUs. FUGIO can be run up to 12 hours for each target application.
- **Publicly available?:** The artifact is released at `https://github.com/WSP-LAB/FUGIO-artifact/tree/v0.1`.

A.3 Description

A.3.1 How to access

Users can access the artifact by cloning the repository from `https://github.com/WSP-LAB/FUGIO-artifact/tree/v0.1`.

A.3.2 Hardware dependencies

Although FUGIO does not require high-performance machines, it is better to have many cores of CPU and large capacities of RAM for parallel fuzzing. Note that we performed the experiments on a machine equipped with 88 cores of CPUs and 384 GB of RAM.

The artifact requires less than 5 GB of disk for each Docker container, but it requires more GBs for running FUGIO that stores identified POP chains and generates exploit objects to be fed to

the fuzzer. Depending on the target application, it might require hundreds GBs of disk space.

A.3.3 Software dependencies

We tested the Docker images and scripts on a Ubuntu 18.04 machine. The artifact requires only Docker; thus, it might work on any OS if it supports Docker. Other software packages will be installed using the provided scripts.

A.4 Installation

1. Install Docker and set that you can run docker commands with a non-root user.
2. Set up RabbitMQ by running the script `run_rabbitmq.sh`.
3. For each version of PHP, build Docker image using the script `1_docker_build.sh` and run a Docker container using the scripts `2_docker_run.sh` and `3_docker_exec.sh`.
4. In the Docker container, install dependencies for FUGIO by running the script `install_XX.sh`, depending on the version of PHP.
    - PHP 5.4: `install_54.sh`
    - PHP 5.6: `install_56.sh`
    - PHP 7.2: `install_72.sh`
5. Prepare environment for operating web applications. Start Apache web server and MySQL using the script `start.sh`. Then, make an account of MySQL using the script `create_user.sh`.

* For more details, please refer to the artifact repository.

A.5 Experiment workflow

1. Prepare a target web application. The artifact provides dump files of applications and databases for convenient settings. Install all or each application using the script `install.py`.
2. Add `.htaccess` file for monitoring POI vulnerabilities by running the script `htaccess.py`.
3. Prepare two terminals; one is for running FUGIO and the other is for triggering POI vulnerabilities.
4. In the first terminal, run FUGIO using the script `run_FUGIO_XX.sh` with the path of the target web application's source code.
5. In the other terminal, trigger the corresponding POI vulnerability using the given scripts in the `Trigger` directory.

* For more details, please refer to the artifact repository.

A.6 Evaluation and expected results

In the evaluations in Sections 7.2 and 7.3, we show that

1. FUGIO can automatically generate exploits for identified POP chains with zero false positives
2. FUGIO can generate exploits for some of the POP chains reported by Dahse *et al.*
3. FUGIO can generate new exploits compared to PHPGGC listed

using Tables 1, 2, and 3, respectively.

Table 1 shows that all exploitable chains that FUGIO generated are indeed exploitable chains (the left number of the plus sign in true positive chains). The number of true positive chains in Table 1 is manually analyzed. For Table 2, we could not match each exploitable chain since Dahse *et al.* did not provide the details of each chain. Thus, we compared the numbers of exploit objects that FUGIO reported with the numbers reported in their paper. Table 3 shows that FUGIO reported new 32 exploitable chains that PHPGGC does not list. PHPGGC provides templates for generating POP exploits. However, it is not clear that what POP gadgets each POP chain consists of. Thus, we provide POP chains from PHPGGC in the FUGIO repository (`https://github.com/WSP-LAB/FUGIO`). FUGIO repository also includes a utility for helping the analysis of the generated POP chains. For more details, please refer to the artifact repository.

The followings are steps for reproducing Tables 1, 2, and 3. First, please follow the installation step described in A.4. Second, for each target application, follow the instructions described in A.5. It takes much time to reproduce all target applications. We recommend selecting target applications that take less time and produce many exploits. When FUGIO finished analyzing the target application, FUGIO generates a dump file of summaries, which is stored in the directory `Files/dump_files`. When triggering a POI vulnerability of the target application using the given script, crawler, or other tools, FUGIO generates a PUT, which is stored in the directory `Files/fuzzing/[APP_PATH.TIME]/PUT`. When FUGIO identifies a POP chain, it saves it as a file in the same directory of the PUT file. If a POP chain reaches the sensitive sink, FUGIO reports the POP chain as a probably exploitable chain. If the POP chain invokes the sink with a parameter containing the attack payload, FUGIO reports the POP chain as an exploitable chain. Such explot objects are saved in the `PROBABLY_EXPLOITABLE` and `EXPLOITABLE` directories, respectively. FUGIO also prints the number of the identified POP chains, probably exploitable chains, and exploitable chains to the console. The results can be compared with Tables 1, 2, and 3 in the paper.