USENIX Security '25 Artifact Appendix: FIXX: FInding eXploits from eXamples

Neil P Thimmaiah
University of Illinois Chicago
npemma2@uic.edu

Yashashvi J Dave
University of Illinois Chicago
yashashvidave@gmail.com

Rigel Gjomemo
University of Illinois Chicago
rgjome1@uic.edu

V.N. Venkatakrishnan
University of Illinois Chicago
venkat@uic.edu

A Artifact Appendix

This artifact appendix is meant to be a self-contained document which describes a roadmap for the evaluation of our novel approach FIXX. It includes a description containing all the software requirements that should be met before proceeding to run FIXX. This appendix also contains details on how to set up your environment to run FIXX.

A.1 Abstract

FInding eXploits from eXamples (FIXX) is a novel approach focused on detecting taint-style vulnerabilities in PHP applications. Using important information from a CVE containing details regarding a vulnerability in an application, our method aims to discover possible similar vulnerabilities in the corresponding application. This artifact contains details on how FIXX can be used to analyze web applications and detect vulnerable paths similar to a previously known vulnerable path. It contains all the requirements needed to set up the environment, as well as details on how FIXX can be installed. Finally, the major claims of the paper have been discussed, and detailed experiments on how these claims can be reproduced.

A.2 Description & Requirements

In this section, we describe the artifact used to run FIXX as well as discuss the requirements needed to run the artifact. Our approach, FIXX, can detect similar exploitable paths in a web application that are similar to a previously known vulnerable path in the corresponding application. FIXX is primarily written in Python and requires certain libraries and packages to perform its analysis. We have discussed each of these dependencies in the subsections below.

A.2.1 Security, privacy, and ethical concerns

FIXX does not pose any security or data privacy concerns when it's being used to discover exploitable paths in a given application. A minor concern: during the primary prerequisite step of FIXX's analysis, which involves reproducing the exploit wherever possible for applications to be analyzed, the evaluators should not use personal information when attempting to create accounts during the installation of applications.

A.2.2 How to access

Our artifact can be accessed using the following URL:
https://doi.org/10.5281/zenodo.14738531

A.2.3 Hardware dependencies

There are no hardware dependencies required to run FIXX.

A.2.4 Software dependencies

The artifact required to run FIXX does require certain software packages and libraries to be installed to perform the analysis. The most important thing to note here is the artifact should be run inside a docker container. To make it easier, we have provided a docker image that contains all the requirements needed to create a new docker container that supports the analysis. We have outlined all the requirements in a docker file called docker-compose.yaml as well as provided instructions on how to retrieve our docker image from Docker Hub in the subsection Set Up. We have also included all the packages in the respective files of the code so that there is no additional error while installing FIXX. Finally, in order to fine-tune the GPT4 model with your own custom key, the library openai needs to be installed.

A.2.5 Benchmarks

FIXX doesn't require any dataset to run the primary results presented in the paper. However, to obtain the precise metrics of the GPT4 model reported in our paper, the evaluators are required to create a personalized GPT4 model and fine-tune it with 50 CVE descriptions. While we have provided the model ID used to obtain the results presented in the paper, OpenAI prevents the sharing of API keys and disables them when they are uploaded to digital repositories. However, we have provided a dataset of 50 unique CVE descriptions that the evaluators can use to fine-tune their custom GPT4 model. Once that is complete, the authors can use any of the CVEs reported in Table 2 of our paper to test their model and reproduce the metrics reported in the paper.

Additionally, FIXX requires the request data as well as the malicious payload that is sent as input to every application when performing the analysis. We have provided a detailed table within our artifact that contains the corresponding files for each of the applications discussed in our paper. Lastly, since FIXX detects similar vulnerabilities to a known vulnerability, while not mandatory for all applications, the exploit must be thoroughly reproduced in order for FIXX to be able to discover accurately the other similar paths in the application. This could be inputting the malicious payload into the vulnerable parameter, inserting the payload into the database, etc. We have provided a detailed example within our artifact pertaining to this.

A.3 Set-up

The following steps detail all the requirements needed to set up your environment to run FIXX:

Docker Image : Obtain the image neildocker693/fixx from Docker Hub. The image can be pulled using the command: **docker pull neildocker693/fixx**. This image should have all the requirements required to run FIXX.

Docker Container : Once the image has been pulled, you can build the container using the command: **docker-compose up -d fixx**. This will create and start the container. You can verify that the container is running using the command: **docker ps**

Docker SSH : Finally, you can ssh into the created container using the command: **docker exec -it fixx /bash/bin**

Please note that if you encounter an error associated with the neo4j database not being available, check the corresponding database name that was loaded and make sure it matches the name of the application you are trying to process. Please refer to the README provided in the zip file of FIXX for troubleshooting such errors.

A.3.1 Installation

Once the artifact has been downloaded from the URL provided in Section A.2.2, unzip and extract its contents into the /opt/project/FIXX/ directory specified inside the docker container.

A.3.2 Basic Test

To test whether the docker container has been set up according to the requirements, you can run the command: **python fixx_main.py -n <application_name> –loadcpg**, which will load the code property graph (CPG) of the corresponding application. Note that if you are analyzing an application outside of the ones provided within our artifact, you will need to build the CPG of the application. This can be done using the command: **python fixx_main.py -n <application_name> –buildcpg**. Once the CPG of the application has been loaded, please refer to the README file within our artifact for a few intermediate steps before analyzing the applications using FIXX.

A.4 Evaluation workflow

We represent the main claims made by the paper and outline the steps needed to be followed to reproduce the results to validate those claims in the following subsections:

A.4.1 Major Claims

In this section, we provide the major claims made in our paper, along with the corresponding experiments that can done to reproduce the results of these claims.

(C1): FIXX can extract the most reused and sensitive instructions in a given PHP application, referred to as seeds, by computing their average reusability scores, sensitivity scores as well as selection scores. This is proven by the experiment (E1) in Section 5 whose results are described in column 1, 2 and 3 of Table 2 in the paper.

(C2): FIXX discovers multiple similar nodes at a specified similarity modulo for the maximum number of seeds (1–4) for each of the selected PHP applications. This is proven by the experiment (E2) in Section 5 whose results are described in column 4 of Table 3.

(C3): FIXX detects multiple similar paths (*sPaths* and *dPaths*) at the same similarity modulo value for each of the obtained similar nodes at a specified similarity threshold for the selected PHP applications. This is proven by the experiment (E3) in Section 5 whose results are described in column 5 of Table 2 as well as column 5 and column 6 of Table 3 respectively.

(C4): FIXX detects multiple new exploitable paths, 10 of which have been reported to MITRE and published as new CVEs. This is proven by the experiment (E4) in Section 5 whose results are described in column 7 and column 9 of Table 3.

(C5): FIXX can rediscover the original exploitable path described in the vulnerability of a given application by

using the newfound *sPaths* and *dPaths*. This is proven by the experiment (E5) in Section 5 whose results are described in column 8 of Table 3.

A.4.2 Experiments

In this section, we provide a detailed set of experiments that can help reproduce the results reported in the paper and support the claims made in the previous section. We also provide the running times for each of the experiments. Please note that since our approach has been evaluated on 19 unique PHP CVEs, we have provided an approximate compute time and human time for all the applications combined, which also includes a bit of manual work when switching between applications described in the CVEs. Finally, for any reported initial CVE in the paper, we recommend verifying all the claims associated with that CVE before proceeding with another CVE to save time and effort.

(E1): [Seed Scores] [*2 computed hours + 1 human hour*]: This experiment will aim to obtain the maximum number of seeds (1–4) for each of the applications as well as compute their reusability, sensitivity and selection scores as described in Table 2. By reproducing these results, the claim C1 can be verified.

How to: To run this experiment, you will need the command **python fixx_main.py -n <application_name> –seedanalysis -vulnerablefile <vulnerable_file_name> -seeds 4**. Note that, as mentioned in the paper, certain applications have limited seeds, so you will need to reduce the seeds for a few of them accordingly. The maximum number of seeds for each of the applications can be found in column 5 of Table 2.

Preparation: To prepare FIXX to perform this analysis for a given application, we need to load the corresponding Code Property Graph (CPG) of the application in the docker container. This can be done using the command **python fixx_main.py -n <application_name> –loadcpg**. Next, make sure to obtain the execution trace of the vulnerable file according to the CVE description of that application. Please refer to the provided README on how to gather the execution trace before analyzing an application. Once the CPG has been loaded and the execution trace has been collected, you can proceed to execute the experiment.

Execution: Run the command provided in the **How to**, which will compute the total number of seeds, i.e., 4, which was used to evaluate the applications in our paper, as well the average reusability scores, sensitivity scores and selection scores for each of them. Note that you are free to evaluate any of the applications on any number of seeds. The higher the number of seeds, the more instructions, although less sensitive, will be considered when computing similar paths

Results: Once FIXX has extracted the seeds and computed the corresponding scores for them, the results will be stored in the /results/seed_results.txt file. This file will contain the location for each of the 4 requested seeds in the corresponding application, as well as their reusability, sensitivity, and selection scores. Finally, the average scores can be found at the end of the file, which can be compared to the results in columns 1, 2, and 3 of Table 2 in our paper.

(E2): [Similar Nodes Detection] [*2 computed hours + 1 human-hour*]: This experiment will aim to obtain unique, similar instructions at a specified similarity modulo for each of the detected seeds.

How to: To run this experiment, you will need the command **python fixx_main.py -n <application_name> –computesimilarnodes -vulnerablefile <vulnerable_file_name> -seeds 4 -modulo 2**.

Preparation: To evaluate our claim, a modulo of 2 can be used. However, FIXX accepts any positive modulo, including 0.

Execution: Run the command provided in the **How to**, which will compute the total number of unique, similar instructions at a modulo 2 for each of the 1–4 seeds, depending on the application, which was used to evaluate the applications in our paper. Note that you are free to evaluate any of the applications on any modulo. The higher the modulo, the lesser the similarity between instructions being considered by FIXX

Results: The sim_nodes_results.txt file inside the results folder can be viewed to verify the different similar instructions along wth their location in the application that have been detected by FIXX. These results correspond to column 4 in Table 3 of our paper.

(E3): [Path Detection] [*15 compute-hour + 10-12 human-hours*]: This experiment will aim to obtain the *sPaths* and *dPaths* reported in column 5 of Table 2 and column 5, 6 of Table 3.

How to: To run this experiment, you will need the command **python fixx_main.py -n <application_name> –detectsimilarpaths -vulnerablefile <vulnerable_file_name> -seeds 4 -modulo 2 -threshold 50**.

Preparation: To evaluate our claim, a threshold value of 50 must be used. However, FIXX accepts any threshold values between 0 and 100 (both included)

Execution: Run the command provided in the **How to**, which will compute the total number of *sPaths* and *dPaths* at a similarity modulo of 2 and a threshold of 50 for each of the 1–4 seeds and detected similar instructions, depending on the application, which was used to evaluate the applications in our paper. Note that this step can take a while, depending on the size of the application. For the extremely large applications SeoPanel and Collabtive, we have written additional code in the fixx_similar_exploits_analyzer.py file. Just search for

SeoPanel or Collabtive, depending on the application that you are testing, and uncomment the corresponding block of code. This will make the processing time of FIXX even faster due to optimal cypher queries that have been written for these applications.

Results: The *sPaths* and *dPaths* per seed can be verified from the terminal and correspond to column 5 of Table 2 in the paper. The path_results.txt file inside the results folder can be viewed to verify the *sPaths* and *dPaths* that have been detected by FIXX. This file contains each of the complete paths along with the total number of *sPaths* and *dPaths* which correspond to column 5 and column 6 in Table 3 of our paper.

(E4): [Exploitable Paths] [*10 human hours*]: Note that this experiment is based on the results obtained from the previous experiment. E3 returns a list of the *sPaths* and *dPaths*. It also prints the total number of exploitable paths in the same text file. However, as mentioned in the paper, in addition to the paths detected by FIXX, we verify the exploitable paths manually. So, we check each of the paths returned by FIXX in E3 and verify the total number of exploitable paths, which correspond to Column 7 of Table 3 in our paper. An example of this is shown in the Example README file.

(E5): [Re-Discover CVE] [*20 compute-hour + 10-12 human-hours*]: For this experiment, we have listed the running time for all the applications combined. This experiment will aim to obtain the original exploitable path reported in the CVE using the newfound *sPaths* and *dPaths*. Note that this experiment can take a lot of time as it repeats the entire analysis of FIXX, and we only report the result as a Yes or No in column 8 of Table 3 in the paper, which verifies if the original path was found or not. We have provided a thorough example in our Example.md file

How to: To run this experiment, you will need the command **python fixx_main.py -n <application_name> –detectsimilarpaths -vulnerablefile <vulnerable_file_name> -seeds 4 -modulo 2 -threshold 50**. Please note that the value of the threshold will need to be altered accordingly since certain paths cannot be rediscovered with the same threshold due to limitations in terms of ancestor/descendant nodes or data flow paths, as discussed in our paper.

Preparation: To evaluate this claim, you will need to select any one of the obtained *sPaths* and *dPaths* and obtain the exploit string as well as the request_data.json file.

Execution: Before running the command provided in the **How to**, you will need to obtain the execution trace of the file in which the selected *sPath* or *dPath* is present. Once the trace has been obtained, you can run the above command to repeat the analysis of FIXX. The path results will be stored in the same folder and can be verified to check if the original vulnerable path is present or not.

Results: The path_results.txt file inside the results folder can be viewed to verify the new *sPaths* and *dPaths* that have been detected by FIXX. You can manually verify that the original exploitable path is present among these paths and confirm with the results shown in column 8 of Table 3.

A.5 Notes on Reusability

Our artifact can be used to detect all exploitable paths in any PHP web application similar to a previously known vulnerable path present in that application. You can modify the values of the seeds, modulo as well as the threshold parameter to discover different types of exploitable paths. Please refer to our paper to avoid resulting in false positive cases by increasing the value of the seeds and similarity modulo or decreasing the value of the similarity threshold parameters, respectively. Significantly increasing the seed value or modulo value can lead to FIXX selecting instructions that are not quite sensitive or differ quite a bit in terms of the identifiers involved. Similarly, decreasing the threshold significantly can lead to FIXX selecting paths that are not similar to the original path. Although FIXX has been evaluated on vulnerabilities like Cross-Site Scripting and SQL Injection, the artifact can discover paths associated with other vulnerability types like Command Injection, etc., as long as the right request is sent when obtaining the execution trace, as this is the basis for discovering similar instructions that lead to similar paths.

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/.