USENIX Security '24 Artifact Appendix: Dancer in the Dark: Synthesizing and Evaluating Polyglots for Blind Cross-Site Scripting

Robin Kirchner*[†], Jonas Möller[‡], Marius Musch[†], David Klein[†], Konrad Rieck[‡], Martin Johns[†]

[†] Technische Universität Braunschweig, Germany
[‡] Technische Universität Berlin, Germany

{robin.kirchner, m.musch, david.klein, m.johns}@tu-braunschweig.de
{jonas.moeller.1, rieck}@tu-berlin.de

A Artifact Appendix

This artifact appendix is meant to be a self-contained document that describes a roadmap for evaluating our artifact.

A.1 Abstract

In "Dancer in the Dark: Synthesizing and Evaluating Polyglots for Blind Cross-Site Scripting", we introduced a cross-site scripting polyglot synthesis approach based on Monte Carlo tree search (MCTS). To achieve full coverage of our large XSS testbed, based on the Google Firing Range (GFR), we designed a generation approach that synthesizes a set of complementing polyglots.

We have linked our public GitHub repository containing our code, usage documentation, and troubleshooting. The code includes instructions on generating polyglots with our approaches, minimizing a polyglot, and testing polyglots on the GFR. The repository includes our small and large testbed, tokens, minimization, and the implementations of all generation approaches we discuss in Appendix C. To maximize compatibility, we have containerized the implementation in a Docker setup. Furthermore, we have added a detailed discussion of our large GFR-based testbed in Appendix D, explaining which tests were excluded and why. We have refined our wording regarding the Minimization process in the main paper, Section 3, thereby clarifying it.

A.2 Description & Requirements

To facilitate cross-platform support, our artifact is containerized with Docker and Docker Compose.

A.2.1 Security, privacy, and ethical concerns

There is no security or privacy risk to be expected when running our artifact. The generation and testing of XSS polyglots happen on the client side in isolated Docker containers. Utilizing the polyglots you synthesized for any other purposes exceeds the scope of our research and this artifact. As for all security testing techniques, we generally advise against applying the polyglots outside of a controlled environment.

A.2.2 How to access

Our artifact is available on our project's GitHub page https://github.com/polyxss/bxss/tree/4f5f2d1db0480c84f20206066ccf09afd937a307. Please use `git clone` or download the project as a zip file. The URL of the corresponding commit for this artifact submission is https://github.com/polyxss/bxss/commit/4f5f2d1db0480c84f20206066ccf09afd937a307.

A.2.3 Hardware dependencies

Docker reports requiring at least 4 GB of RAM, a 64-bit kernel, and CPU support for virtualization.

Recommendation: In our repository, see Section A.2.2, we have prepared a compose file for the generation process. With the given parameters, it takes approximately 5 hours to conclude a polyglot set generation with Docker confined to 8 GB RAM, 8 CPU cores, and 30 GB of free disk space.

A.2.4 Software dependencies

Our artifact uses Docker and Docker Compose, which are available for macOS, Linux, and Windows. **If Docker is installed on your machine, you may skip this section.**

Our software was tested extensively on macOS and used productively on Linux-based servers. Below, we compile the starting points for installing Docker and Docker Compose on macOS and Ubuntu Linux. Detailed instructions for these and all other supported operating systems are available on Docker's homepage https://docs.docker.com/engine/install/.

*Corresponding author

macOS Docker supports the most recent version of macOS, plus the previous two releases.

macOS with Apple silicon If your Mac runs an Apple Silicon chip, e.g., M1, M2, or M3 MacBooks, download the Docker `dmg` image directly from Docker. Double-click the image to start the installation process. It includes the Docker desktop GUI and Docker Compose so that you can continue with the next steps.

macOS with Intel chip If your Mac runs an Intel chip, e.g., 2019, 2020 MacBooks, download the `Docker.dmg` image from Docker. Double-click the image to start the installation process. It includes the Docker desktop GUI and Docker Compose so that you can continue with the next steps.

Ubuntu Linux For Ubuntu, please follow the installation instructions from the docker docs. For Linux-based systems, the software requirements entail:

- KVM virtualization support,
- QEMU version 5.2 or later,
- Gnome, KDE, or MATE Desktop environment.
- Refer to Docker's complete list of requirements.

A.2.5 Benchmarks

None.

A.3 Set-up

Once you have Docker and Docker Compose installed, you are ready to go. See Section A.3.1 to learn how to download the repository and build the project.

A.3.1 Installation

1 a) Installing our project is as easy as cloning a Git repository. See Figure 1 for the corresponding bash code.

```
#!/bin/bash
git clone git@github.com:polyxss/bxss.git
cd bxss
mkdir -p data/out/runs
git reset --hard e8c22be
```

Figure 1: Clone our repository, move into its directory, create the output folder, and switch to the specific commit.

1 b) If you don't have Git installed, download the repository as a `.zip` file, unpack it, and `cd` into it. See Figure 2 for the corresponding commands.

2) You can build the project with Docker Compose. Use our build script for this task. Building should be fast, but it depends on your download speed.

```
#!/bin/bash
# Define the repository URL and the commit hash
REPO_URL="https://github.com/polyxss/bxss"
COMMIT_HASH="4f5f2d1db0480c84f20206066ccf09afd937a307"

# Download the commit as a zip file
curl -L "${REPO_URL}/archive/${COMMIT_HASH}.zip" -o commit.zip

# Unzip the downloaded file
unzip commit.zip

# Change directory into the unzipped folder
cd "bxss-${COMMIT_HASH}"
```

Figure 2: Download our repository as a zip, unpack it, and move into it.

```
#!/bin/bash
./scripts/build.sh
```

A.3.2 Basic Test

Now that you have downloaded and built our repository, you can start with the functionality tests. We prepared an easy-to-use starting script in the scripts directory. Call the scripts from the project's root directory (where you should be currently in). For a first basic test, use our `./scripts/1-basic-test.sh` script. This will generate one single polyglot in approximately a minute. The current run's output directory is to `data/out/runs/run-<timestamp>/`. In the run's output directory, a `run-<timestamp>/summary/` folder will be created, containing the final polyglot set, which consists of one polyglot in this test.

```
#!/bin/bash
# Quickly create a first polyglot
./scripts/1-basic-test.sh
```

Note that the actual runtime depends on your available resources and allocated to Docker. This first test is limited in its purpose of demonstrating the approach. Our synthesis approach runs MCTS, which inherently has a random component. The polyglot you are generating is built step by step and changes (improves) over time. Intermediary steps and the final step are logged in the run's output directory under `run-<timestamp>/try-1/best-polyglot/`. A finished synthesis creates a `final-polyglots.json` file in `run-<timestamp>/summary/`. See Figure 3 for the data directory structure.

To evaluate how good the quickly created polyglot already is, run the next test and select the `run-<timestamp>` folder corresponding to your run, i.e., likely the most recent timestamp.

Please confirm that you have at least one final polyglot in the output directory before continuing with the next steps.

```
#!/bin/bash
# Evaluate the created polyglot; follow the
 ↪ CLI instructions and select the timestamp
 ↪ of your run
./scripts/5-eval.sh
```

```
data
├── example.json
├── gfr_tests.csv
└── out
    └── runs
        ├── example-run
        │   └── summary
        │       ├── final-polyglots.json
        │       └── final-polyglots0-eval.json
        └── run-2023-08-28T13-49-11.354Z
            ├── meta.json
            ├── summary
            │   ├── final-polyglots.json
            │   ├── final-polyglots0-eval.json
            │   └── final-polyglots1-eval.json
            └── try-1
                └── best-polyglot
                    ├── bestOutput-0000000000
                    ├── bestOutput-0000000050
                    ├── ...
                    ├── bestOutput-0000000450
                    └── bestOutput-00000final
```

Figure 3: The data directory.

A.4 Evaluation workflow

Now that you have generated a basic polyglot, you could allocate more time to the generation process. We have prepared four configurations (1) and estimated the required time on a machine with 8GB RAM and 8 CPU threads.

We prepared an evaluation script (2) that runs polyglots from the `data/out/runs/*` directory against the full range of relevant Google Firing Range (GF) tests. The evaluation then returns a score.

A.4.1 Polyglot Generation

You can try out the different scripts if you want to spend the required time. We recommend letting the basic test (a) and maybe the extended test (b) finish generating polyglot. Thus, your new polyglots will be evaluated in Section A.4.2.

(a) **Basic Test**

- Approximate runtime one minute
- Generates a proof-of-concept polyglot
- `./scripts/1-basic-test.sh`

(b) **Extended Test**

- Approximate runtime 15 minutes
- Generated a single polyglot
- `./scripts/2-extended-test.sh`

(c) **Long Test***

- Approximate runtime 5 hours
- Generated a useable single polyglot
- `./scripts/3-long-test.sh`

(d) **Polyglot Set***

- Approximate runtime multiple days (2-5 days)
- Generates a polyglot set
- `./scripts/4-set-generation.sh`

** Please note that the long test and polyglot set generation process can take significantly longer than we calculated.*

A.4.2 Polyglot Evaluation

Now that you have generated polyglots on the fast testbed, you can evaluate them on the GFR. The test runs a polyglot against the GFR tests and counts a test as solved if the polyglot manages to print `xss` to the console.

- Approximate runtime <2 minutes
- Evaluates a polyglot against the GFR and prints a score
- `./scripts/5-eval.sh`

A.4.3 Cleanup

You may call this script once you have completed the entire artifact evaluation. It will stop the Docker services.

- Approximate runtime <2 minutes
- Stops the corresponding docker services of this project.
- `./scripts/6-stop.sh`

A.4.4 Major Claims

(C1): We propose a method for automatically synthesizing a small set of XSS polyglots using Monte Carlo tree search. Allocating sufficient time, this can be reproduced with Experiment (E1). We show the general approach in Algorithm 1 of the "Dancer in the Dark" paper and publish our code with this artifact.

(C2): We show that our polyglots cover all currently known injection contexts. *Allocating sufficient time, this can be reproduced with Experiment (E1) using the script (4). However, for the sake of this artifact, it is sufficient to spend less time on the polyglot generation, which should* *still generate one usable polyglot.* The results from our polyglots are displayed in Figure 4 of the “Dancer in the Dark” paper.

Please note that, due to ethical reasons, our synthesized polyglots are not public. However, we open-sourced our generation technique to facilitate making your own.

A.4.5 Experiments

Here, we provide instructions for an experimental generation of a polyglot and a polyglot set.

(E1): *[Polyglot Generation] [5 human-minutes + 15 compute-minutes]:*
How to: Follow Section A.4.1(b) for the approximately 15-minute generation task. Once finished, a polyglot is written to the run's summary directory. This step can be replaced by A.4.1(a-d), depending on the time allocated for this evaluation. *Even the shortest generation, A.4.1(a), will work for the next step.*
Preparation: After following the installation instructions from Section A.3, no further preparation is required. Per default, the `data/run/` directory exists. Make sure it still exists.
Execution: Run `./scripts/2-extended-test.sh` or another of the scripts in Section A.4.1.
Results: A `final-polyglots.json` fill will be created under `run-<timestamp>/summary/`. See Figure 4 for an example.

```
{
  "data": [
    {
      "id": 0,
      "payload": "<sVg;;<ScRiPt
       ↪  sRc=`http://testbed:8080/xss.js`>....)"
    }
  ]
}
```

Figure 4: Example for final-polyglots.json

(E2): *[Polyglot Evaluation] [10 human-minutes + 5 compute-minutes]: ...*
How to: Follow Section A.4.2 to evaluate your newly created polyglot on the GFR.
Preparation: After following the installation instructions from Section A.3, no further preparation is required. Per default, the `data/run/` directory exists. Make sure it still exists.
Execution: Run `./scripts/5-eval.sh`. Follow the instructions in the console and select the corresponding run.
Then, wait for the evaluation to finish.

```
$ ./scripts/5-eval.sh
Please select a run to evaluate:
1. example-run
2. run-2023-08-28T13-49-11.354Z
3. run-2024-06-13T12-58-44-635Z
4. run-2024-06-17T16:52:15.414Z
Enter the number of your choice: 1
```

Results: Once finished, the score will be printed to the console.

```
Enter the number of your choice: 1
Reading
 ↪  ...example-run/summary/final-polyglots.json
Writing to ...example-run/summary/

[+] Running 3/3
runner-1  | Successful tests: 65/175
runner-1  | Done
```

An extended evaluation is in a file next to the `final-polyglots.json` file in your run's summary folder.

A.5 Notes on Reusability

Our generation process is flexible, as Section 7 of the paper indicates. The provided inputs influence the generated polyglots. This includes the tests of the test bed and the token set. Changing both can shift the generation's focus to other areas beyond the scope of our work. For instance, tests can be narrowed down to a specific vulnerability or expanded to support specific web frameworks.

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/usenixsec2024/.