USENIX'23 Artifact Appendix: Greenhouse - Single-Service Rehosting of Linux-Based Firmware Binaries in User-Space Emulation

Hui Jun Tay, Kyle Zeng, Jayakrishna Menon Vadayath, Arvind S Raj, Audrey Dutcher, Tejesh Reddy, Wil Gibbs, Zion Leonahenahe Basque, Fangzhou Dong, Zack Smith, Adam Doupé, Tiffany Bao, Yan Shoshitaishvili, Ruoyu Wang

Arizona State University

A Artifact Appendix

A.1 Abstract

The primary artifact provided is a Docker image containing an implementation of the Greenhouse prototype presented in our paper, along with the complete dataset of 7,140 firmware images referenced in our evaluation. Greenhouse is a Python3 framework that implements *user-space single-service rehosting*, using various *interventions* detailed in our paper to enable the emulation of a specific web-facing service for a given firmware image. Unlike previous rehosting works, the rehosted firmware service is executed via user-space emulation (qemu-user) instead of a full-system emulation environment.

We evaluated Greenhouse on a dataset of 7,140 firmware images from nine different vendors to demonstrate its scalability and generalizability. Greenhouse successfully rehosts 2,841 HTTP web-services, and an additional 685 web-services to partial connectivity. Our experiment demonstrates the usability of these images by finding 717 N-day vulnerabilities using the open-source framework Routersploit and 26 zero-day vulnerabilities through fuzzing with AFL++.

A.2 Description & Requirements

Greenhouse was evaluated using a kubernetes cluster containing 42 nodes and over 2,000 CPU cores in order to complete our analysis on 7,140 firmware images. The Docker image packaged in this artifact contains an entrypoint script that the cluster pods use to run Greenhouse on each firmware target. As this script is designed for automation with our kubernetes cluster setup, we have also provided a *run.sh* wrapper script for the purposes of manual evaluation. Our submitted artifacts consist of the following items:

- *greenhouse-ae.tar*, a prepackaged Docker image containing our experiment setup for manual evaluation
- *greenhouse-rehosted.csv*, a file detailing the rehosting success of each sample in our dataset
- *gh2routersploit.csv*, a file mapping the 717 N-days found to their respective rehosted targets
- source code and instructions for building the Greenhouse docker image, fuzzer component and minikube setup on GitHub

Within the prepackaged Docker container are the following:

- a standalone version of Greenhouse for manual evaluation + a run script */gh/run.sh*
- a modified routersploit framework used to find the 717 N-days on our rehosted images + a run script */routersploit/run_routersploit.sh*
- 2 crashing input files that demonstrate two of the 26 zero-day vulnerabilities discussed in our paper that have since been publicly released by D-Link

A.2.1 Security, privacy, and ethical concerns

Greenhouse itself has a low risk of causing issues on the machine it is run on. However, many of its functions require control of device mounts and network interfaces that necessitate that the Docker image is run in *privileged* mode. While Greenhouse itself does not perform any malicious activity, the firmware it is emulating may execute commands that affect the host machine. It is thus recommended to run Greenhouse within the provided container to minimize the impact of such behavior on the host machine.

A.2.2 How to access

A copy of our artifact is available on Zenodo (https://doi.org/10.5281/zenodo.8217895). We also open-sourced the code used to build our Greenhouse artifacts on GitHub[1].

[1] https://github.com/sefcom/greenhouse/tree/08f7caf456f31de4f9c25325302705f7881a5e39

A.2.3 Hardware dependencies

Greenhouse requires at least 1 CPU core and 8GB of RAM. As the amount of storage needed varies based on the firmware and scale of the job, we recommend having at least 50GB of disk space available. Our full dataset of all 7,140 firmware images requires at least 125GB of disk space. The total disk space of our experiment, including all rehosted images and log files, is approximately 655GB. To perform large-scale evaluation of Greenhouse, a kubernetes cluster is necessary. Running on a local minikube instance is possible, but is unstable compared to kubernetes and may have reduced rehosting performance.

A.2.4 Software dependencies

Greenhouse was tested on a host machine using Ubuntu 20.04 and Python 3.7. Greenhouse is dependent on *qemu-user, docker, angr* and *binwalk*, and makes use of another rehosting tool, *FirmAE*, as a supplementary component. The Docker image provided as part of the artifact contains a stable, working version of Greenhouse for evaluation and all third-party software needed for it to run. The artifact Docker image provided must be run in *privileged* mode for optimal results. It is recommended that the host machine be running Ubuntu 20.04 or later, and have Docker and *docker-compose* installed.

A.2.5 Benchmarks

Greenhouse was run on a dataset of 7,140 firmware images crawled from nine different vendors. This dataset is hosted privately as part of our artifact submission. Please contact the authors for access to the dataset if necessary.

A.3 Set-up

Install Docker and *docker-compose* on the host machine.

- *Docker version 24.0.2, build cb74dfc*
- *docker-compose version 1.29.2, build 5becea4c*

To manually validate the rehosted images, we recommend installing *curl* and a web-browser such as Firefox.

Greenhouse and Docker use the network ip addresses in the range 172.17.0.0 and 172.21.0.0 by default. We recommend keeping these network ranges open. A significant number of firmware web services were observed to make use of addresses in the range 192.168.0.0. Thus, we recommended ensuring that ip addresses in this range are available during the rehosting and testing process.

A.3.1 Installation

- Load the Docker image with *'docker load -i greenhouse-ae.tar'*
- Check that the image *greenhouse:usenix-eval-jul2023* is present *'docker image list -a'*
- Start the container in privileged, interactive mode:

  docker run --privileged -v /dev:/host/dev -it greenhouse:usenix-eval-jul2023 bash
- Copy a firmware image file from the dataset into the Docker container:

  docker cp <externalpathtoimage> <container-name>:/<imagepath>
- Inside the Docker container, run the setup script:

  bash /gh/docker_init.sh
- The container and target are now ready.

A.3.2 Basic Test

The provided Docker image comes with a simple bash script */gh/test.sh* that can be run from within the Docker container. One run, the script should exit with the message 'All tests pass!' after about a minute of execution.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *Greenhouse is able to perform user-space rehosting of single-services with a success rate of 39.7%. This is proven by experiments described in Section 7.2 of our paper. The results of our evaluation are reported in Table 2 of our paper, and reflected in the greenhouse-rehosted.csv table provided as part of this artifact evaluation. Note that due to the non-deterministic nature of rehosting firmware, the exact number of rehosted services may fluctuate, but should average out to our experimental numbers over a sufficiently large dataset.*

(C2): *Firmware images rehosted by Greenhouse are of sufficient fidelity to be used with dynamic analysis to find real-world vulnerabilities. Greenhouse found 717 N-day vulnerabilities on images it rehosted using routersploit, and 26 zero-day vulnerabilities via fuzzing with AFL++. This is described in Section 7.4 and 7.5 of our paper, with results specified in Table 7 for the routersploit N-days and Table 9 for the crashing inputs found*[2]*. A breakdown of these numbers is reflected in the routersploit.csv table provided.*

[2]We do not provide all the crashing inputs mentioned in Table 9 as not all have been made public. The 2 crashing inputs provided with the artifact were publicized here: https://supportannouncement.us.dlink.com/announcement/publication.aspx?name=SAP10313

A.4.2 Experiments

(E1): [Rehosting a single firmware] *[30 human-minutes 2 to 8 compute-hours + 50GB disk]*: This section describes how to run Greenhouse to rehost a single firmware image using the provided Docker image artifact. This experiment validates C1.

Preparation: Make sure to setup the Docker container provided and the target sample as discussed in the Installation section above.

Execution: To rehost a target firmware image with Greenhouse, run

/gh/run.sh <brand> <image-path-in-container>

from inside the Docker container. The script performs the rehosting from start to finish, printing logs to *stdout* and */tmp/gh.logs*. A step-by-step can be found in the README file with the rest of our Zenodo artifact. A larger scale, parallelised approach to running Greenhouse can be done with a kubernetes cluster or minikube setup. Instructions on how to do so can be found in the MINIKUBE.md file on our GitHub.

Results: Greenhouse takes approximately 2 to 8 hours to rehost an image. When it completes, the script will print 'GHREHOST COMPLETE'. The rehosted image itself can be found inside the container under */gh/results/<sha256hash>*.

The file *config.json* describes the results of the rehosting. A 'SUCCESS' result corresponds to the Interact column of Table 2, which contributes to our total of 2,841 rehosted images. More detailed instructions on how to manually run and evaluate an individual rehosted image are in the README file.

Note that the Greenhouse rehosted services may superficially deviate from the original, though functionality is usually unaffected. As emulating firmware images tends to come with a degree of non-determinism, results observed may also vary on a sample-to-sample basis. This should average out over larger sets for a given brand.

(E2): [Exploit replay with routersploit] *[30 human-minutes + 4 compute-hours + 50GB disk]*: This section describes how to use the rehosted image created in E1 with routersploit and crashing scripts to validate C2.

Preparation: Make sure that a rehosted image is available inside the container with a folder named debug (e.g. */gh/results/<sha256hash>/debug.*)

Execution: Routersploit can be run inside the Docker artifact via a script given the path to a rehosted image (e.g. */gh/results/<sha256hash>*.)

/routersploit/run_routersploit.sh <path-to-rehosted-image>

Results: The script takes approximately 4 hours to run all 125 N-days that are built into the routersploit framework used for our evaluation. Results should be automatically consolidated inside */routersploit/results/vulnerable.csv*. 'gh2routersploit.csv' is a breakdown mapping each routersploit N-day to the rehosted sample on which it found the vulnerability.

(E3): [Validating crashing PoCs] *[10 human-minutes + 0.2 compute-hours + 50GB disk]*: This section describes how to use the two crashing inputs provided on their corresponding Greenhouse rehosted services to validate C2. Only two of the 26 vulnerabilities discovered are provided as the rest have to yet to be made public at the time of this report.

Preparation: Make sure that a rehosted image is available inside the container with a folder named debug (e.g. */gh/results/<sha256hash>/debug.*) The crashing input files are available inside the folder */crashing_inputs*.

Execution: Follow instructions in the README on starting up a Greenhouse rehosted image manually using docker-compose. Once the rehosted firmware is fully up, emit the crashing input to its target using the built-in netcat client:

cat <crashing-input-file> | nc -w2 <ip> <port>

Results: The two crashing inputs provided are for two zero-days found for the *DIR-601_REVA_1.02* and *DIR-825_REVB_2.03* firmware samples in Table 9 and Table 11 of our paper. Emitting them to the rehosted web service should cause it to crash, with a segmentation fault in the terminal output of the artifact container.

A.5 Notes on Reusability

Greenhouse can be extended to run on other types of web-services. We provide two such extensions in the image for UPNP and DNS services.

A.6 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/.