Your Firmware Has Arrived: A Study of Firmware Update Vulnerabilities

Yuhao Wu, Jinwen Wang, Yujie Wang, Shixuan Zhai, and Zihan Li, *Washington University in St. Louis;* Yi He, *Tsinghua University;* Kun Sun, *George Mason University;* Qi Li, *Tsinghua University;* Ning Zhang, *Washington University in St. Louis*

https://www.usenix.org/conference/usenixsecurity24/presentation/wu-yuhao

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

USENIX Security ’24 Artifact Appendix: Your Firmware Has Arrived: A Study of Firmware Update Vulnerabilities

Yuhao Wu[†], Jinwen Wang[†], Yujie Wang[†], Shixuan Zhai[†],
Zihan Li[†], Yi He[§], Kun Sun[‡], Qi Li[§], Ning Zhang[†]

[†] Washington University in St. Louis,
[§] Tsinghua University, [‡] George Mason University

A Artifact Appendix

A.1 Abstract

ChkUp is an approach designed to Check for firmware Update vulnerabilities by extracting program execution paths from a firmware update procedure and identifying vulnerable verification steps within that procedure. Our artifact includes ChkUp’s source code. Additionally, we release lists of the collected firmware images, which can be downloaded and unpacked using our provided scripts. To utilize ChkUp, users need to run our program from the command line, providing the file paths of unpacked firmware images. The expected output comprises vulnerability identification results (i.e., vulnerable verification procedures) and intermediate structural, syntactic, and semantic analysis results for programs related to firmware updates. To streamline the artifact evaluation (AE) process, we provide a pre-configured virtual machine (VM) with all necessary dependencies installed. Remote VM access is available through AnyDesk and TeamViewer.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Conducting the AE for ChkUp does not raise security, privacy, or ethical concerns. All programs and data are securely stored within a VM hosted on a remote server, and the AE activities are carried out inside this VM. This setup ensures that the AE process is isolated so that there is no security and privacy concern associated with using artifacts.

A.2.2 How to access

We have made the source code publicly available on GitHub: https://github.com/WUSTL-CSPL/ChkUp/tree/973a9ecc81a320e0537a4f6625fda8704f0bf7fc.

A.2.3 Hardware dependencies

To assess our artifacts, please ensure that your host system maintains a stable network connection for accessing the remote VM via AnyDesk or TeamViewer. Our VM is equipped with an AMD Ryzen 9 3900X 12-Core Processor, 4GB of RAM, and a 40GB disk capacity. While no additional specific hardware is required, please note that variations in hardware may result in differences in runtime performance.

A.2.4 Software dependencies

ChkUp is primarily developed using Python 3.6.9, with the environment established through Miniconda 23.11.0 on Ubuntu 20.04.6 LTS. The main static analysis tools employed are Angr 9.2.6 and Ghidra 10.1.2. All Python dependencies are listed in the *requirements.txt* file. Additionally, the *README.md* file in the GitHub repository provides detailed software installation guidance.

A.2.5 Benchmarks

We have released lists of the collected firmware images on a GitHub repository: https://github.com/WUSTL-CSPL/Firmware-Dataset. Additionally, we provide Python scripts to help with downloading (*fw_downloader.py*) and unpacking (*fw_unpacker.py*) firmware images.

A.3 Set-up

Note: Reviewers can skip this section. We provide a fully configured remote VM to simplify the artifact evaluation process, accessible through AnyDesk and TeamViewer. As a result, the only setup required is installing AnyDesk or TeamViewer; no additional test environment is needed. This section is provided solely to document our VM setup process.

A.3.1 Installation (Optional)

Linux packages: The necessary Linux packages for setting up ChkUp can be installed by using the following commands:

```
$ sudo apt install git npm net-tools
$ sudo apt-get install openjdk-11-jdk
```

Python dependencies: Miniconda is the recommended tool for setting up the environment. It is available for download from the official website at https://docs.conda.io/en/latest/miniconda.html. It can be installed by following the instructions provided on the website. Once installed, use the commands below to establish the Python environment:

```
$ conda create --name chkup python=3.6.9
$ conda activate chkup
$ cd <repository_path>
$ pip install -r requirements.txt
```

Npm modules: There are some npm modules required for ChkUp, which can be installed through:

```
$ cd <repository_path/exec_pth_rec/jsparse>
$ npm install
```

Ghidra: The release file of the reverse engineering tool Ghidra is available from its repository at https://github.com/NationalSecurityAgency/ghidra/releases/download/Ghidra_10.1.2_build/ghidra_10.1.2_PUBLIC_20220125.zip. Once downloaded and extracted, it works with the installed JDK.

A.3.2 Basic Test

The successful setup of a Python environment can be verified with the following commands:

```
$ conda --version
$ python --version
```

To check the installation of npm, execute:

```
$ npm --version
```

Furthermore, to ensure the JDK is operational for supporting Ghidra, use the following command:

```
$ java -version
```

If the final outputs do not indicate any errors, it can be concluded that these dependencies are installed properly.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): ChkUp can identify firmware update programs within an image and uncover their execution flows.

(C2): ChkUp can identify verification procedures within a firmware update process.

(C3): ChkUp is capable of detecting vulnerabilities in the verification procedures of a firmware update process.

A.4.2 Experiments

(E1): *[Execution Path Recovery] [3 human-minutes + 14 compute-minutes*]:

Preparation: The provided VM is fully configured (see set-up details in Section A.3) and three firmware samples have been downloaded and unpacked in the VM (using the *fw_downloader.py* and *fw_unpacker.py* scripts as mentioned in Section A.2.5). Thus, there is no additional step to prepare for this experiment. We provide a script to run ChkUp with the three firmware samples for analyzing their firmware update procedures.

Execution: The experiments can be conducted using the *chkup_run.sh*, which runs the *main.py* script for each sample firmware image by specifying the firmware path and result storing path. To run these experiments, use the following commands:

```
$ conda activate chkup
$ cd <repository_path>
$ ./chkup_run.sh
```

Note that in our configured VM, <*repository_path*> is set to */home/chkup/Desktop/ChkUp*.

Results: The analysis results for each firmware sample will be printed out, while detailed analysis results are stored in the directory <*repository_path/results/firmware_name/exec_pth*>. Specifically, for each firmware sample, the update flow graph (i.e., UFG, stored as *ufg-<entry_program_name>.pkl*) and control flow graphs (i.e., CFGs, stored as <*program_name*>*.pkl*) are formatted as NetworkX graphs. The *results.json* file under the result folder offers a more explainable representation of results, containing key details about the execution paths and the connections between firmware update-related programs. The JSON structure starts with a top-level key that specifies the entry program's file type, followed by its file path as the second-level key. Subsequently, it includes nested program entries that detail all related programs and their interactions. Each entry includes the program type, associated files (caller and callee), interaction methods, and additional evidence of their involvement in firmware updates. Additionally, the pattern-matching results for each type of program are stored in individual JSON files under the result folder. For example, *webinfo.json* stores the pattern-matching results of related web programs.

(E2): *[Verification Procedure Recognition] [3 human-minutes + 7 compute-minutes]:*

Preparation: The script *chkup_run.sh* contains verification procedure recognition process.

Execution: After executing *chkup_run.sh*, verification procedure recognition should run automatically following the execution path recovery.

Results: The verification procedure recognition results can be found from the printed output and <*reposi-*

tory_path/results/firmware_name/ver_proc> directory. Under the directory, *results.json* contains JSON structure where keys indicate the program path and values are identified verification function pairs (target function name: identified function address) in the firmware update procedure. There are also intermediate results for identifying verification procedures stored under the result directory. Specifically, the constructed data flow graphs (i.e., DFGs, in NetworkX graph format) are stored in the *dfg* folder and the function similarity scores are stored in the *vuln_corpus_config* folder.

(E3): *[Vulnerability Discovery] [3 human-minutes + 3 compute-minutes*]:

Preparation: The script *chkup_run.sh* contains vulnerability discovery process.

Execution: After executing *chkup_run.sh*, vulnerability discovery should run automatically following the verification procedure recognition.

Results: Besides the printed output, the vulnerability discovery result folder <*repository_path/results/firmware_name/vuln_discov*> contains two files named *results.json* and *vuln_discov.log*. These files include detailed information on the identified vulnerabilities for each firmware image, including the vulnerability category and identified vulnerable verification procedure details. For instance, the *results.json* file contains three keys, *Proper*, *Improper*, and *Missing*, where the details of proper, improper, and missing verification procedures are listed.

A.5 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/.