Pandawan: Quantifying Progress in Linux-based Firmware Rehosting

Ioannis Angelakopoulos, Gianluca Stringhini, and Manuel Egele, *Boston University*

https://www.usenix.org/conference/usenixsecurity24/presentation/angelakopoulos

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: Pandawan: Quantifying Progress in Linux-based Firmware Rehosting

Ioannis Angelakopoulos, Gianluca Stringhini, and Manuel Egele
Boston University
{jaggel, gian, megele}@bu.edu

A Artifact Appendix

A.1 Abstract

Our artifacts include the source code and instructions about how to install the Pandawan prototype and use it to holistically re-host and analyze the code of IoT firmware images. We provide two examples of firmware images that can be analyzed with Pandawan. We also packaged Pandawan within a docker image along with the three re-hosting systems (FirmSolo, Firmadyne, and FirmAE) and the PyPANDA dynamic analysis system that we used to demonstrate Pandawan's utility. The docker image can run on any system that has docker installed.

In this appendix, we describe the steps to analyze a sample firmware image using Pandawan. The analysis process involves reverse engineering the original firmware kernel, invoking Pandawan's `Kernel Augmentation` technique and building an "augmented" custom kernel that is capable of holistically analyzing IoT firmware code.

Finally, we demonstrate how Pandawan's holistic re-hosting and analysis can be used to analyze the IoT kernel modules within firmware images and how Pandawan's `FICD` technique can be used to compare different re-hosting systems on their re-hosting capabilities.

A.2 Description & Requirements

A.2.1 How to access

You can access the artifacts at https://github.com/BUseclab/Pandawan/tree/v1.0.0

A.2.2 Hardware dependencies

None

A.2.3 Software dependencies

Python, Docker and Java (for Ghidra).

A.2.4 Benchmarks

For the artifact evaluation we use an example Netgear firmware image as a benchmark ($id = 1$).

A.3 Set-up

A.3.1 Installation

Using the Docker:

Download the docker image from:
https://doi.org/10.5281/zenodo.7865451
Install the docker image:

```
docker load < firmsolo.tar.gz
git clone https://github.com/BUseclab/Pandawan.git
cd Pandawan
docker build -t pandawan .
```

Spawn a docker container:

```
docker run -v $(pwd):/output --rm -it
--privileged pandawan /bin/bash
```

Manual Installation:

To manually install and run Pandawan you first need to install these dependencies:

```
<install_dir>/Pandawan/install.sh
<install_dir>
```

where `<install_dir>` is the directory where Pandawan is installed (/ inside the container).

Install Ghidra:

Follow instuctions in https://ghidra-sre.org/InstallationGuide.html

Note: The `install.sh` script will make use of legacy (and unavailable) compiler toolchains that are currently only installed within the Docker image. Also we use a modified version of FirmAE and Firmadyne (to be compatible with Pandawan) for our experiments. You have to manually modify the scripts in these systems to specify the installation directories of Firmadyne and FirmAE (not required if you use the Docker container).

Set Toolchains:

```
cd <install_dir>/FirmSolo
```

Here follow the instructions in https://github.com/BUseclab/FirmSolo.git to setup the installation directories and toolchains that will be used by FirmSolo.

Set paths:

```
cd <install_dir>/Pandawan
```

Modify the `paths.py` script and properly set the directories specified in the script.

A.3.2 Basic Test

Our basic test for Pandawan includes gathering metadata for a firmware image. Please proceed as follows (assuming you are using the Docker container):

Spawn a Docker container according to Section A.3.1 and run:

```
mkdir -p /output/images/
```

Download the example images and store them in the output directory:

```
git clone \
https://github.com/BUseclab/FirmSolo-data.git

mv ./FirmSolo-data/images/* /output/images/
```

In this case `/output/` is your work directory.

Then execute:

```
cd /Pandawan

python3 run_pandawan.py 1 -t s1

ls /output/pandwan_results/Image_Info/
```

After running the `ls` command you should see a file named `1.pkl` within the `/output/pandawan_results/Image_Info/` directory.

Note: The basic test might take a long time (around 2 hours) due to the static analysis (the mapping of kernel symbols to configuration options) happening in the background. Increase the `num_of_threads` variable within the `custom_utils.py` script within FirmSolo's directory to speed up the experiment.

A.4 Evaluation workflow

A.4.1 Major Claims

Pandawan is a framework that holistically (both the user and kernel level) re-hosts and analyzes Linux-based IoT firmware code. Furthermore, the `FICD` technique, implemented in Pandawan, enables the comparison of different re-hosting systems based on the re-hosting capabilities. Below we list and prove the claims related to the evaluation of our artifact.

(C1): *Pandawan reverse engineers the original kernel of a firmware image and builds an "augmented" custom kernel supported by QEMU that can holistically re-host and analyze a firmware image. This claim is proven by experiment (E1), described in Section 5.3 in our paper.*

(C2): *Downstream analysis systems can use Pandawan's holistic analysis results to analyze the binary firmware kernel modules for bugs and vulnerabilities. This claim is proven in Experiment (E2), described in Section 5.4 and Table 2 in our paper.*

(C3): *The `FICD` technique can be used to compare different re-hosting frameworks on their re-hosting capabilities. This claim is proven in Experiment (E3), described in Section 5.5 and Table 3 in our paper.*

A.4.2 Experiments

We assume that you use the docker image to perform the artifact evaluation.

(E1): *[Reverse Engineering] [5 human-minutes + 30 compute-minutes + 5GB disk]: In this experiment Pandawan reverse engineers the original firmware kernel and uses the `Kernel Augmentation` technique to produce an "augmented" custom kernel conducive to holistic re-hosting and analysis. Pandawan also proceeds to create a dedicated PyPANDA script for the firmware image and holistically re-host it using PyPANDA.*

Preparation: *Copy the extracted file-system and kernel of the target firmware image in the work directory.*

Execution: *After you install and connect to the docker container as described in Section A.3.1, proceed to analyze an example firmware image:*

Within the docker execute:

```
mkdir -p /output/images/
```

On your host download the images from this link (if you have not implemented the basic test):

```
git clone \
https://github.com/BUseclab/FirmSolo-data.git
```

Execute:

```
mv ./FirmSolo-data/images/* /output/images/
```

In this case `/output/` is your work directory.
To analyze image 1 with Pandawan, inside your container execute:

```
cd /Pandawan/
python3 run_pandawan.py 1 -a -s -g 2700 -p
"\-f 300 \-s \-t \-c"
```

Pandawan will analyze firmware image 1, reverse engineer the original firmware kernel and build a new kernel that is capable of holistically re-hosting and analyzing image 1. Pandawan will create a dedicated PyPANDA script for 1 located in `/output/pandawan_results/scratch/1/run.py` and use that script to holistically re-host image 1 with PyPANDA.

Results: *When the emulation concludes you*

should be able to find the serial log output (`qemu.final.serial.log`), the recorded tasks (i.e., executed programs with their arguments) (`proc_logs.txt`), and other information relative to the re-hosting process in within:

```
/output/pandawan_results/scratch/1/
```

Note: Depending on the metadata information extracted/processed about the original firmware kernel and the fidelity of the re-hosting (if kernel modules crash during the re-hosting process) this experiment might take longer. However, Pandawan implements caching features during each experiment, which renders future runs faster.

(E2): *[TriforceAFL] [1 human-minute + 1.5 compute-hour]: In this experiment you use the results from Pandawan's holistic analysis and the TriforceAFL kernel fuzzer to analyze the kernel modules within the target firmware image.*
Preparation: *None*
Execution: `/Pandawan/fuzzing/fuzzing.sh /Pandawan/ 1 30`
This script will use TriforceAFL and the holistic analysis results of (E1) to fuzz the kernel modules within image 1 for 30 minutes.
The `fuzzing.sh` script will first run the `get_kernel_traces.py` script. This script will translate Pandawan's holistic analysis results (`exec_context.pkl` and `exec_trace.pkl`) stored within `/output/pandawan_results/scratch/1/` into seeds that can be used by TriforceAFL to analyze the firmware kernel modules within image 1. Specifically, the seeds consist of chains of system calls (see Table 6 in our paper), traced along with their arguments by Pandawan during the re-hosting experiment in E1, that lead to the execution of kernel module code. In this example, the `get_kernel_traces.py` will create seeds for the `statistics.ko` kernel module within image 1. Next, `fuzzing.sh` will invoke TriforceAFL (/Pandawan/fuzzing/triforce_run.py) to analyze the above kernel module.
Results: *The fuzzing results will be available in the `/output/Fuzz_Results_Cur/1` directory.*

(E3): *[Comparison Results] [1 human-minute + 120 compute-minutes]: In this experiment you use the `FICD` technique to compare Pandawan, FirmSolo, Firmadyne, and FirmAE across the user programs executed, unique user QEMU Translation Blocks (TBs) executed, kernel modules loaded, and unique kernel module TBs executed metrics.*
Preparation: *None*
Execution: *Run the re-hosting experiments for each re-hosting system as follows:*

```
python3 run_pandawan.py 1 -a -s -g 2700 -p
"\-f 300 \-s \-t \-c" (already executed in E1)

python3 run_pandawan.py 1 -a -s -f -g 2700
-p "\-f 300 \-s \-t \-c"

python3 run_pandawan.py 1 -a -s -e -g 2700
-p "\-f 300 \-s \-t \-c"

python3 run_pandawan.py 1 -a -s -d -g 2700
-p "\-f 300 \-s \-t \-c"

python3 run_pandawan.py 1 -c
```

The first four commands above will run the re-hosting experiments for all the the compared re-hosting systems (Pandawan, FirmSolo, Firmadyne, and FirmAE) while using the `FICD` technique and will collect the metrics described previously. The fifth command will output the metrics for each system so that a comparison between the systems is possible.

Results: *The results will be printed in standard output. You should see the below output:*

```
Metrics for image 1 across the compared
re-hosting systems
System: pandawan
Executed programs: 23, Unique BBs: 11463
Total KOs: 75, Loaded KOs: 35, Persistent
crashes: 0
Unique KO TBs: 881
System: firmsolo
Executed programs: 23, Unique BBs: 11510
Total KOs: 75, Loaded KOs: 35, Persistent
crashes: 0
Unique KO TBs: 787
System: firmae
Executed programs: 23, Unique BBs: 11862
System firmae does not load KOs
System firmae does not execute KO code
System: firmadyne
Executed programs: 23, Unique BBs: 11209
System firmadyne does not load KOs
System firmadyne does not execute KO code
```

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/.