Operation Mango: Scalable Discovery of Taint-Style Vulnerabilities in Binary Firmware Services

Wil Gibbs, Arvind S Raj, Jayakrishna Menon Vadayath, Hui Jun Tay, Justin Miller, Akshay Ajayan, Zion Leonahenahe Basque, Audrey Dutcher, and Fangzhou Dong, *Arizona State University;* Xavier Maso, *unaffiliated;* Giovanni Vigna and Christopher Kruegel, *UC Santa Barbara;* Adam Doupé, Yan Shoshitaishvili, and Ruoyu Wang, *Arizona State University*

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

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: Operation Mango: Scalable Discovery of Taint-Style Vulnerabilities in Binary Firmware Services

Wil Gibbs*, Arvind S Raj*, Jayakrishna Menon Vadayath*, Hui Jun Tay*, Justin Miller*, Akshay Ajayan*
Zion Leonahenahe Basque*, Audrey Dutcher*, Fangzhou Dong*, Xavier Maso†,
Giovanni Vigna‡, Christopher Kruegel‡, Adam Doupé*, Yan Shoshitaishvili*, Ruoyu Wang*

*Arizona State University
{wfgibbs, arvindsraj, jvadayat, htay2, jmill, aajayan, zbasque, dutcher, fdong12, doupe, yans, fishw}@asu.edu
‡University of California, Santa Barbara
{vigna, chris}@cs.ucsb.edu
†contact@xaviermaso.com

A Artifact Appendix

A.1 Abstract

In this paper, we propose a scaling static analysis of firmware binaries so that *all* binaries can be analyzed for command injection or buffer overflows. To achieve this, we developed MANGODFA, a novel binary data-flow analysis leveraging value analysis and data dependency analysis on binary code. Through key algorithmic optimizations in MANGODFA, our prototype Mango achieves fast analysis without sacrificing precision. Mango was tested on the Karonte dataset of 49 firmware to show its performance comparability to SaTC, which we also ran on this dataset. Mango was also tested on the 7 firmware that were handpicked by the SaTC authors to demonstrate the versatility of Mango. We performed an ablation study demonstrating the performance gains in Mango come from key algorithmic improvements. Finally, we evaluated the scalability of Mango on 1700 firmware from the Greenhouse dataset.

A.2 Description & Requirements

Mango can be run parallelized locally with the use of docker containers or with Kubernetes remotely. However, we will not give any support for running on Kubernetes, nor can we provide access to our computing resources. Mango will spit out many results, but for the paper, we only care about results with a score of 7 or higher. This score is pre-pended to the file name of every result and can easily be aggregated with the *find* command.

A.2.1 Security, privacy, and ethical concerns

There are no risks to the evaluators while executing our artifact. However, it should be noted that the output of our artifact will contain potential vulnerabilities and should not be shared with the general public.

A.2.2 How to access

Our artifacts can be accessed on our GitHub[1].

A.2.3 Hardware dependencies

There are no specific hardware dependencies, but we suggest a sufficiently powerful machine to run the experiments. Our experiments were run on a Kubernetes cluster with access to 4000 cores. We also require at least 500GB of storage if you run all experiments. The large dataset experiments consume up to 400GB of disk space.

A.2.4 Software dependencies

Our artifact is consolidated into two Python packages, which can be pip-installed along with their supporting packages. Our artifact is functional on Ubuntu 22.04 x64, using Python 3.11 and a pip version of 24 or greater.

A.2.5 Benchmarks

Here are all of the datasets we used.

karonte dataset: https://drive.google.com/file/d/1-VOf-tEpu4LIgyDyZr7bBZCDK-K2DHaj/view?usp=sharing

[1] https://github.com/sefcom/operation-mango-public/blob/ff15727d3d9f7016e91e3f07a983e81090a62b3d

7 Firmware dataset: https://www.dropbox.com/scl/fi/v7k3c2ech20l5hh6dxhdf/7_firmware.tar.gz?rlkey=lo74bd7wcgf1jmxmxlwv6j4v3&dl=0
ablation dataset: https://www.dropbox.com/scl/fi/zz04neglh9d4ycsqa2c0j/ablation-firmware.tar.gz?rlkey=hwhf9tt21f7hsuzcrbrjkg08t&dl=0
greenhouse dataset: https://www.dropbox.com/scl/fi/2ndob4flx6sn3a53fln83/large_dataset.tar.gz?rlkey=frgjlhwh244mqb4ua1om9atqd&dl=0
additional experiment dataset: https://www.dropbox.com/scl/fi/girnzmyjaterigijmls87/additional_experiment.tar.gz?rlkey=4okv6ivbsyer6df0nc70tzz73&dl=0

A.3 Set-up

You will need to install Python 3.11 or later and pip version 24 or later on your machine. You will also need either Docker or Kubernetes to run the experiments. We also advise installing the artifact packages in a Python virtual , using Python 3.11 and a pip version of 24 or greaterenvironment.

A.3.1 Installation

First, git clone the repository. Next, cd into the repository and run the following commands:
pip install -e .
cd pipeline
pip install -e .

This will install the base package 'argument_resolver' and the 'mango_pipeline' package which is used to run the experiments.

A.3.2 Basic Test

To run a simple functionality test, navigate to the root of the git repo and run the following commands:
pip install pytest pytest-cov
pytest

This will run the tests for the base package and all of the tests should pass.

A.4 Evaluation workflow

All of the reproduction steps are also available on our GitHub[1].

A.4.1 Major Claims

(C1): *Operation Mango achieves a greater amount of vulnerabilities found than prior workflow in a similar amount of analysis time spent. This is proven by the comparison experiment (E1) described in section 11.3 on the Karonte Dataset illustrated in table 1.*
(C2): *Operation Mango finds more vulnerabilities by analyzing more binaries than prior work (E2) in the same time or less as shown in section 11.4 and table 3.*
(C3): *Operation Mango's core contributions of assumed non-impact and sink-to-source analysis are proved in the ablation experiment (E3) from section 11.5 and illustrated in table 5.*
(C4): *Operation Mango is extremely scalable. Section 11.6 performs a large-scale evaluation (E4) across 1700 firmware, whose results are found in table 6.*

A.4.2 Experiments

Please note that we cannot provide support for running any of the experiments on Kubernetes or using any of the compute resources we used for the paper.
(E1): *[Karonte Dataset Comparison] [10 human-minutes + 930 compute-hour + 5GB disk + 10GB RAM]: 49 firmware are used to evaluate the efficacy of Operation Mango's analysis compared to prior work.*
Preparation: *Have both docker and python3.11 installed in a Ubuntu 22.04 environment. Download the karonte-dataset.tgz and unpack it.*
How to: *Please refer to the Table 1 replication steps on our github*[1].
Results: *Run the show_table.py script from our GitHub*[1] *to interpret the results.*
Depending on your compute power, your run-time may be longer than in Table 1, but the amount of TruPoCs and binaries analyzed should be similar.
To differentiate between command injection and overflow results, use the –status flag on mango-pipeline.
(E2): *[7 Manual Firmware Analysis] [10 human-minutes + 8 compute-hour + 5GB disk + 10GB RAM]: In this experiment, we analyzed 7 firmware from the SaTC Paper*[2] *and showed we found vulnerabilities in binaries they did not analyze.*
Preparation: *Have both docker and python3.11 installed in a Ubuntu 22.04 environment. Download the 7_firmware.tar.gz and unpack it.*
How to: *Please refer to the Table 3 replication steps on our github*[1].
Results: *Run the show_table.py script from our github*[1] *with the flag –show-firmware to interpret the results. The results should be similar to those in Table 3 of the paper, but depending on the combination of computer power and timeout length, the results may differ slightly.*
(E3): *[Ablation Study] [10 human-minutes + 3 compute-hour + 1GB disk + 10GB RAM]: In this experiment, we analyze the R6400V2 firmware, turning on and off dif-*

[2] https://www.usenix.org/system/files/sec21fall-chen-libo.pdf

ferent combinations of assumed nonimpact and sink-to-source analysis to show the impact of our contributions. The results should be similar to Table 5 in the paper, but depending on the combination of computer power and timeout length, the results may differ slightly.
Preparation: *Have both docker and python3.11 installed in a Ubuntu 22.04 environment. Download the ablation-firmware.tar.gz and unpack it.*
How to: *Please refer to the Table 5 replication steps on our github*[1].
Results: *Run the ablation.py script from our github*[1] *to interpret the results.*

(E4): *[Large Scale Analysis] [10 human-minutes + 8157 compute-hour + 400GB disk + 10GB RAM]: In this experiment, we analyze 1700 firmware from the greenhouse dataset from a wide variety of different vendors. The results should be similar to those in Table 6 of the paper, but depending on the combination of computer power and timeout length, the results may differ slightly.*
Preparation: *Have both docker and python3.11 installed in a Ubuntu 22.04 environment. Download the large_dataset.tar.gz and unpack it.*
How to: *Please refer to the Table 5 replication steps on our github*[1].
Results: *Run the show_table.py script from our github*[1] *to interpret the results.*

A.5 Notes on Reusability

Operation Mango is built upon angr and can be extended for other static analysis tasks. The base *mango* can be used for non-firmware static analysis out of the box.

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/.