USENIX'23 Artifact Appendix: ARI: Attestation of Real-time Mission Execution Integrity

Jinwen Wang[*], Yujie Wang[*], Ao Li[*], Yang Xiao[†],
Ruide Zhang[‡], Wenjing Lou[‡], Y. Thomas Hou [‡], Ning Zhang[*]

[*] Washington University in St. Louis
[†] University of Kentucky
[‡] Virginia Polytechnic Institute and State University

A Artifact Appendix

A.1 Abstract

This artifact includes all the source code of Attestation of Real-time Mission Execution Integrity (ARI), a policy-guided real-time mission execution integrity attestation framework. It mainly contains three key component. A compartmentalization mechanism, runtime mission information measurement mechanism, and a mission integrity verification engine. For this artifact evaluation, we will illustrate ARI 's functionality by employing an example policy to verify the execution integrity of a copter flight mission. Specifically, the policy will utilize controller-based compartmentalization, with the attitude controller as the critical compartment. We aim to streamline the Artifact Evaluation (AE) process by providing a pre-configured virtual machine (VM) with all necessary dependencies. Additionally, we provide a hardware flight controller accessible via SSH. For further convenience, remote VM access is made available through Teamviewer.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Carrying out the Artifact Evaluation (AE) for ARI does not pose any security, privacy, or ethical concerns. All AE tasks are carried out within the Virtual Machine (VM) and the remote hardware flight controller, which are host on remote machine. This ensures that the AE process remains isolated and does not interact with the reviewer's personal or sensitive code/data.

A.2.2 How to access

Source Code: https://github.com/WUSTL-CSPL/ARI

A.2.3 Hardware dependencies

To properly evaluate our artifact, please ensure that your host system has stable network to access the remote VM through Teamviewer. Furthermore, the quadcopter mission runs in a simulation environment on a Raspberry Pi 3, equipped with a Navio2 daughter board. To access the remote Raspberry Pi 3 with the Navio2 daughter board, you can utilize the following command and corresponding password in remote VM:

A.2.4 Software dependencies

To undertake an effective evaluation of the ARI artifact, it is recommended to use the Ubuntu 16.04 LTS operating system. Our customized compiler is built upon LLVM 3.9.0, while the software quadcopter flight controller utilizes ArduPilot 3.9.0. The underlying verification engine has been developed based on Capstone 4.0.2, and employs the Black2s hash algorithm. The cross compiler is gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf. The OS on Raspberry Pi3 is Linux navio 4.14.95-emlid-v7+.

A.2.5 Benchmarks

None

A.3 Set-up

The setup section is intended for reviewers who wish to construct the system from scratch. We have made available a well-configured remote VM accessible via Teamviewer. Thus, reviewer can skip section 3.

A.3.1 Installation

Dependencies Installation: Install the dependencies for the three key components of ARI by using the following commands.

```
$ sudo apt-get install python-pip
$ python -m pip install –upgrade "pip < 19.2"
$ sudo python -m pip install –upgrade "pip < 21.0"
$ sudo apt-get install clang-3.9 && cd /usr/bin && sudo ln ./clang-3.9 ./clang && sudo ln ./clang++-3.9 ./clang++
$ sudo apt install git cmake build-essential make texinfo bison flex ninja-build ncurses-dev texlive-full binutils-dev python-networkx python-matplotlib python-pygraphviz python-serial
$ sudo pip2 -q install -U future lxml pymavlink MAVProxy
$ pip install pydotplus python-louvain bitarray capstone enum34 pyelftools pyblake2
$ wget http://launchpadlibrarian.net/356067403/gcc-5-aarch64-linux-gnu_5.4.0-6ubuntu1~16.04.9cross1_amd64.deb
$ sudo dpkg -i ./gcc-5-aarch64-linux-gnu_5.4.0-6ubuntu1 ~16.04.9cross1_amd64.deb
```

Customized LLVM Installation: *ari_dir* is the root directory of ARI project. In VM *ari_dir* is */home/ari-new-ae/conattest*

```
$ git clone https://github.com/WUSTL-CSPL/ARI.git
$ cd ./conattestllvm
$ chmod +x ./compiler_for_1st_part.sh
$ ./compiler_for_1st_part.sh
$ mkdir build && cd ./build
$ cmake -DLLVM_ENABLE_ASSERTIONS=OFF ..
$ make
$ echo 'export PATH=$PATH:ari_dir/
conattestllvm/build/bin' ≫ ~/.bashrc
$ source ~/.bashrc
```

ArduPilot Installation:

```
$ echo 'export PATH=$PATH:ari_dir/
gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-
gnueabihf/bin' » ~/.bashrc
$ source ~/.bashrc
```

Download the Pi3 image in following link. Decompress it into *pi3_img_dir*.https://drive.google.com/drive/folders/1WOiFES-zJf6JkdWjziMnFrqsJJlmlBwy?usp=sharing.

```
$ cd pi3_img_dir/my-working-image && ./load_image.sh
```

A.3.2 Basic Test

You can verify the success of the LLVM installation by using the command *llvm-config –version*. A successful installation will return *3.9.0svn* as the result

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): ARI is capable of compartmentalizing and instrumenting the system in accordance with the policy.

(C2): ARI can automatically instrument the CPS, recording control flow events during runtime.

(C3): ARI has the ability to verify mission integrity based on runtime measurements.

A.4.2 Experiments

(E1): [Automatic Compartmentalization and Runtime Measurement Instrumentation] [5 human-minutes + 30 compute-minutes]

How to: Given CPS software such as ArduPilot, ARI uses command line instructions to automatically compartmentalize and instrument the software.

Preparation: Implement the compartmentalization policy in *ari_dir/graph_analysis/analyze.py* (*ari_dir* refers to the root directory of ARI , i.e., */home/ari-new-ae/conattest in VM*). We have provided the default one, *partition_by_controller*. Identify the critical compartments in *ari_dir/ardupilot/crit_cpt*; we have designated the attitude controller as the critical compartment. **Please note that the terminal might display a 'Build Failed' message due to linking issues during the ArduPilot build, but this is normal; we will link it later.**

Execution:

```
$ cd ari_dir
$ cd ./conattestllvm && ./compiler_for_1st_part.sh
$ cd ./build && make -j2
$ cd ../../ardupilot
$ source ./compile_1st_part.txt
$ cd ../conattestllvm && ./compiler_for_2nd_part.sh
$ cd ./build && make -j2
$ cd ../../ardupilot
$ source ./compile_2nd_part.txt
```

Results: The resulting binary is stored in *ardu_dir/build/sitl/bin/arducopter* and is also transferred to the Pi3. You can check the compartmentalization of the application into 8 regions with the following command:

```
$ readelf -S ardu_dir/ardupilot/build/sitl/bin/arducopter
```

(E2): [Mission Execution and Measurement Collecting] [5 human-minutes + 10 compute-minutes]:

How to: Execute a takeoff mission with Ardupilot on Pi3. The instrumented code will automatically record the runtime measurements.

Preparation: (1) Log into the remote Pi3 via SSH from VM. (2) Open a terminal in the VM.

Execution: (1) On the remote Pi3, execute the following commands:

```
$ ssh pi@10.228.106.170
$ cd ~&& sync
$ sudo ./arducopter -S -I0 –model + –speedup 1 –defaults ./copter.parm
```

(2) In the VM, run the following single line command:

```
$ "mavproxy.py" "–master" "tcp:10.228.106.170:5760" "–sitl" "10.228.106.170:5501" "–out" "10.228.106.170:14550" "–out" "10.228.106.170:14551" "–map" "–console"
```

You will then see a *Console* and a *Map* in the VM. Wait for approximately 1 minute until the *Console* displays *AP: EKF2 IMU0 is using GPS* and *AP: EKF2 IMU1 is using GPS*.

Next, type the following commands in the terminal (only the command after > in the following):
STABILIZE> mode guided
GUIDED> arm throttle
GUIDED> takeoff 20

After the flight takes off (wait for about 10 seconds), stop the processes in the VM terminal and remote Pi3 using *Ctrl + C*. Then, transfer the measurement from the remote Pi3 to the VM terminal using these commands:
$ cd ardu_path/ardupilot/
$ source ./tsf_measurement.txt

Results: The mission measurements are stored in *ardu_path.txt*, including *ARI_branch.txt*, *ARI_ind_jmp.txt*, *ARI_ret_hash.txt*, and *ARI_tsf.txt*.

(E3): [Measurement Verification] [1 human-minutes + 3 compute-minutes]
How to: Perform the verification using runtime measurements as input. The verification engine will issue an alert if the verification fails.
Preparation: Open a terminal in the VM.
Execution: *$ cd ../oat-verify-engine*
$ source ../ardupilot/mission_verify.txt

Results: The verification engine will validate the mission integrity. If the verification passes, it will display the hash of all return addresses obtained from both runtime and replay. A successful verification will also show *Return Integrity Verification Pass!*

A.5 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/`.