USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Software Availability Protection in Cyber-Physical Systems

Ao Li, Jinwen Wang, and Ning Zhang, *Washington University in St. Louis*

https://www.usenix.org/conference/usenixsecurity25/presentation/li-ao

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

USENIX Security '25 Artifact Appendix: Software Availability Protection in Cyber-Physical Systems

Ao Li*, Jinwen Wang*, Ning Zhang
Computer Security and Privacy Laboratory
Washington University in St. Louis

A Artifact Appendix

A.1 Abstract

The artifact contains the source code of Gecko, an attack recovery approach that not only restores execution promptly after an attack but also disables exploited features to enhance system availability. It comprises three key components: (1) compartmentalization and instrumentation, (2) a shadow compartment mechanism, and (3) checkpoint/restore mechanisms. For this artifact evaluation, we demonstrate Gecko's functionality by showcasing its ability to recover ArduCopter from emulated attacks. Specifically, (1) ArduCopter is compartmentalized using our customized compilers, and (2) it recovers from a triggered crash with the support of the shadow compartment mechanism and checkpoint/restore mechanisms. We aim to streamline the Artifact Evaluation (AE) process by providing a pre-configured machine with all necessary dependencies.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Conducting the Artifact Evaluation (AE) for Gecko does not raise any security, privacy, or ethical concerns. All AE tasks are performed on a dedicated remote machine specifically used for AE, ensuring that the process remains isolated and does not interact with the reviewer's personal or sensitive code/data.

A.2.2 How to access

The source code is available on GitHub at https://github.com/WUSTL-CSPL/Gecko, as well as on Zenodo: https://zenodo.org/records/15049225.

A.2.3 Hardware dependencies

There is no specific hardware dependency. For convenience during evaluation, we provide access to a remote machine set up via TeamViewer.

A.2.4 Software dependencies

Gecko is evaluated using ArduPilot (Copter-4.0) as the target application. The testing environment consists of Ubuntu 20.04 as the operating system and LLVM version 13.0.1 as the compiler.

A.2.5 Benchmarks

None

A.3 Set-up

This setup is intended for users who wish to build the system from scratch. However, to simplify the process, we provide a pre-configured Docker container that allows reviewers to skip Section 3 entirely. The container is available at: ghcr.io/a01ixxx/gecko-image:stable.

A.3.1 Installation

Prerequisite APT packages:

```
$ sudo apt install cmake build-essential
    make texinfo bison flex ninja-build git
    gitk git-gui ncurses-dev texlive-full
    binutils-dev python3-networkx python3-
    matplotlib python3-pygraphviz python3-
    serial python3-pip python3-distutils
    python-is-python3 tmux libgtk-3-dev
    libwebkit2gtk-4.0-dev libjpeg-dev
    libtiff-dev libsdl1.2-dev libgstreamer1
    .0-dev
```

Prerequisite pip packages:

```
pip install community pydot opencv-python
    future MAVProxy wxPython
```

LLVM Installation: Gecko is tested on LLVM-13. User can either install the package on ubuntu or compile from source code. Due to space constraints, please refer to the official instructions for more details.

Build SVF program analysis tool:

```
$ cd /path/to/Gecko/SVF
$ ./build.sh
```

Build CRIU checkpoint/restore tool:

```
# install depdendencies
$ sudo apt install libdrm-dev gnutls-dev
    libnftables-dev libbsd-dev  libprotobuf-
    dev libprotobuf-c-dev protobuf-c-
    compiler protobuf-compiler python3-
    protobuf libnl-3-dev libcap-dev  uuid-
    dev libbsd-dev libnftables-dev libnet1-
    dev gnutls-dev libdrm-dev
$ cd /path/to/Gecko/checkpoint_restore
$ make clean
$ make
```

ArduPilot Installation:

```
$ cd /path/to/Gecko/ardupilot
$ ./waf configure --board sitl
$ ./waf build
```

A.3.2 Basic Test

You can verify the success of the LLVM installation:

```
$ llvm-config --version
13.0.1
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Gecko is capable of compartmentalizing and instrumenting the system while enabling real-time recovery.
(C2): Gecko's recovery capability allows CPS to recover from triggered failures.

A.4.2 Experiments

(E1): This verifies the compartmentalization and instrumentation processes (C1), which are applied at compile time. [5 human-minutes + 15 compute-minutes]

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

```
$ cd ~/ardupilot_redcaps/
$ ./1_compartmentalization.sh
```

Result E1-1: The compartmentalization result is in the file *./build/sitl/compartments_result.json*. The functions and variables are partitioned into different compartments (code and data regions).

Next, Gecko uses the following command line to automatically instrument the defense mechanism in the given application with our customized compiler.

```
$ ./2_instrumentation.sh
```

Result E1-2: The final binary result is the file *./build/sitl/bin/arducopter*. You can use the following command to check the instrumented attack detection codes.

```
objdump -D ./build/sitl/bin/arducopter |
    grep _dfi
```

(E2): This demonstrates Gecko's ability to recover from a crash (C2). [3 human-minutes + 10 compute-minutes]

Launching the ArduPilot simulation requires multiple commands, so we provide three scripts to simplify the process.

The first script launches ArduPilot and checkpoints a program in memory, which will be used later.

```
$ cd ~/ardupilot_recovery
$ ./prepare_checkpoint.sh
# You may need to enter the sudo passwd
```

The second script automatically configures the drone, launches the mission, and opens two panels to display simulation information.

```
$ cd ~/ardupilot_redcaps/
$ ./launch_demo.sh
[Gecko] Using the native block.
[Gecko] Using the native block.
```

Please wait 3-5 minutes for the drone to start executing the mission, as shown in the figure below:

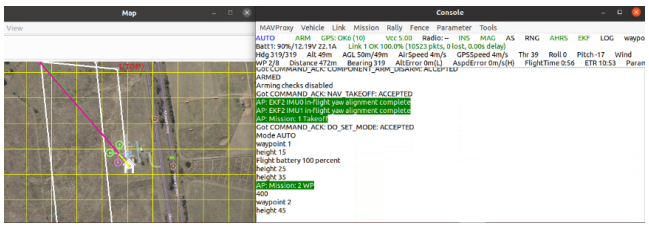

Figure 1: ArduCopter mission example.

Then, return to your main terminal by detaching from the current tmux session: press `Ctrl+b`, then `d`. Next, you can run the thrid script, which is designed to trigger a program crash and then recover it from a checkpoint.

```
$ ./launch_attack_recovery.sh
```

Result E2-1: This will cause the program, namely *arducopter*, to crash and then restore it from a previously saved checkpoint. After recovery, the system transitions to a shadow compartment. This is indicated by continuous logs showing the use of the recovery block, as shown below:

```
[Gecko] The task is restored !!!
With response time: xxx seconds
[Gecko] Using the recovery block.
[Gecko] Using the recovery block.
```

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/usenixsec2025/.