Control Invariant Monitor

This is the user manual for the control invariant monitoring. For more information on the technical details, see the paper "Detecting Attacks Against Robotic Vehicles: A Control Invariant Approach (CCS'18)".

Citation

```
@inproceedings{choi2018detecting,
  title={Detecting attacks against robotic vehicles: A control invariant
approach},
  author={Choi, Hongjun and Lee, Wen-Chuan and Aafer, Yousra and Fei, Fan and Tu,
Zhan and Zhang, Xiangyu and Xu, Dongyan and Deng, Xinyan},
  booktitle={Proceedings of the 2018 ACM SIGSAC Conference on Computer and
Communications Security},
  pages={801--816},
  year={2018}
}
```

Introduction

Control Invariant Monitoring is an attack detection framework on robotic vehicles to identify physical attacks through monitoring **control invariants**. Specifically, the technique extracts such invariants by jointly modeling the physical properties of the system, its control algorithm and the laws of physics.
These invariants are represented in a form of state-space model that can be instrumented to control software and checked at runtime.

In this tutorial, we show that how to extract the invariants with Matlab System Identification Tool and how to detect attacks with the example scenarios.

Setup

Software Requirements

1. Matlab R2017B or above
2. VMware Workstation 12.0 or above
3. VMimage (apmvm 1.0) includes:
    - Ubuntu 16.04.3 LTS
    - ArduCopter V3.4-dev (control software)
    - Mavproxy 1.5.2 (gound control system)
    - SITL (software in the loop) simulator
    - Mission generator
    - Test driver
    - Log converter
    - Matlab codes for SI
    - Monitoring function template in C/C++
    - Output Analyzer

Installation

The provided VM image (id: apmvm / password: apm) includes all the programs and the related packages to test invariant monitoring, expect Matlab tools. For System Identification, Matlab is required to be installed separately from the VM image on your host machine.

VM image are available: https://bit.ly/3o83GQD

Building Control Invariant Model

Building control invariant consists of three main components: control invariant extraction, control software reverse engineering, and monitor code generation.

Invariant Extraction

Data Collection

Our test generation tool can produce random missions. In order to collect profile data, we run a number of missions in simulation.

Running multiple missions automatically

A collection of missions can be executed systematically. Start multiple simulation

```
$ cd ~/ardupilot/Tools/autotest/
$ mission_tools/testdriver.py
```

The default number of random missions is 20. You may change the default value in the test driver script.

The logs files are located in this directory after mission completed.

```
$ ~/ardupilot/Tools/autotest/logs/*.BIN
```

Running a single mission manually (optional)

Besides the automatic missions, you can run a single mission with MAVproxy commands.

Generate multiple random missions:

```
$ cd ~/ardupilot/Tools/autotest
$ mission_tools/mission_generator.py <#missions>

Mission files are located in:
$ ~/ardupilot/Tools/autotest/missions
```

Start SITL simulator:

```
$ ./start_sim.sh
```

Load a mission by entering the following commands in the MAVProxy Command Prompt.

```
STABILIZE> script init.scr
GUIDED> wp load <mission_file>
```

Start a single mission:

```
After "GPS lock" message shown in MAVProxy console,

$ GUIDED> arm throttle
$ GUIDED> takeoff 10
$ GUIDED> mode auto
```

The drone performs the given mission in Auto mode. You will see the flight in the Map View, and the MAVProxy console shows the flight information in details. Once the mission is completed, the drone is disarmed. Then you can finish the simulation.

Finish simulation:

```
ctrl+c
```

System Identification

System Identification creates models from measured input-output data. The collected data need to be preprocessed for SI in Matlab.

Preparing data

Convert the .bin logs to CSV files for MATLAB

```
$ cd ~/ardupilot/Tools/autotest/logs/
$ mavlogdump.py --format csv --types ATT <input> > <output>

Example) for multiple files
$ for i in `seq 1 20`;     do
    mavlogdump.py --format csv --types ATT $i.BIN > "$i"_ATT.csv;
done
```

Move the converted logs to the desired directory so that MATLAB scripts can access (to the Matlab-installed machine).

Place the .csv files into the Matlab working directory

```
"<MATLAB working directory>/logs/"
```

The measured data (especially, real data) often has noises or anomalies. You may view the data, filter out noise, and remove offsets manually. The Matlab tools also provides such functions. For details, refer the Matlab Toolbox manual.

Running system identification (in Matlab)

The Matlab Tools provide various functions for System Identification, including data preprocessing, model generation and model validation. In this example, we provide Matlab command line scripts to extract the model.
Run the Matlab strcipt "system_identification.m", which calculate A,B,C,D matrices. The output message are shown as folloing:

```
====================================
======== State Space Model =========
====================================
A:
        0.9889   -0.0275   -0.0322
    0.0025    1.0000   -0.0000
    0.0000    0.0025    1.0000

B:
    0.0025
    0.0000
    0.0000

C:
    2.7868    7.8921   12.6862

D:
     0
```

The matrices will be populated in the invariant monitor template in C code. The values are required to be recorded for the latter purpose. The generated model is also stored in the Matlab file:

```
ss_model.mat
```

Model Validation (optional)

```
model_validation.m
```

You can test the generated model with a different log that was not used in SI. The log will provide the measurements of input and output. The "model_validation.m" script generates the model response with the given inputs and shows a comparison graph between the model response and the output measurement. You can check how well the model can estimate the actual output with this graph.

Binary Reverse Engineering

In this manual, we skip the binary reverse engineering. Engineering efforts for binary reverse engineering, including manual efforts and some domain-knowledge, would be vary depending on target binaries. In this example, we use a source code to directly insert the extracted invariants, instead of binary rewriting for the illustrative purpose. You can still instrument binaries separately compiled our monitoring function and binary rewriters.

Monitor Generation

Monitoring Parameter Selection

Matlab code:

```
selectparams.m
```

Run the script "selectparams.m". First, the script determines the window size. You may adjust the alpha_w value in the script, which is the margin parameter of the window size. Once the window is decided, the error threshold is hence computed from the maximum observed model-induced errors within the window. To add the margin of the error threshold, adjust the "alpha_th" value (%)

```
====================================
======= Parameter Selection=========
====================================
WindowSize:
       26.4000

ErrorThreshold:
       39.7322
```

Keep the values. The determined parameters will be used in the monitoring function.

Invariants Instrumentation

First, make the complete monitoring function by populating the extracted model and parameters.

The exmaple code is located here:

```
~/ardupilot/ArduCopter/copter_invariant.cpp

//invariant model
static const float    A[3][3] = {{ 0.9884, -0.0493f, -0.0242f}, {0.0025f, 0.9999f,
-0.0000f}, {0.0f, 0.0025f, 1.0f}};
static const float    B[3] = {0.0025f, 0.0f, 0.0f};
static const float    C[3] = {1.8651f, 16.8655f, 10.0631f};
static const float    D = 0.0f;

//monitoring parameters
static const int    window = 1040;      // window size x 40 (26 * 40)
static const float  threshold = 400000;  // threshold x 10000 (40 * 10000)
```

Note that the consistency of the unit of each variable and loop frequency. In our example, the log's sampling rate is 0.1s whereas the rate of the main control loop is 0.0025s. Therefore, the window size should be multiplied by 40. The variable in the log is in degree, but the main loops use the centi-degree. Therefore, the threshold also is multiplied by 10,000.

Once the monitoring function is ready, insert the function call to the main loop. We assume that the user knows the source code of main control loop and the state variable being monitored. We directly insert the monitoring function call at the end of the control loop. For the binaries, please refer our paper.

In our example, the source code of main control loop is located in:

```
$~/ardupilot/ArduCopter/ArduCopter.cpp
```

And at the end of the main control loop, insert the function call

```
copter_invariants_check(<target>, <measured>);
```

For example, the main control loop is

```
void Copter::fast_loop()
```

and inserted monitoring function looks like the following:

```
copter_invariants_check(attitude_control.angle_ef_targets().x, ahrs.roll_sensor);
```

Example usage for a sample attack and detection

This section briefly describes how to launch the sample attack during the mission, and how to detect the attack. The detection results can be observed with an attack alert at runtime and also logs provide the internal changes of the state variable, monitoring status (model response and errors).

Attack and Detection

How to launch attacks

Run a mission: As shown in the data collection section, run a single mission in the simulation.

Launch an attack: The example attack simulates sensor spoofing attack. Our attack simulation code to the interface between the control software and sensor modules and compromise sensor measurements by injecting malicious singles. In the example, the attack module overwrites the roll measurement to the 30 degree.

During the mission, open new terminal, launch the attack module (another GCS) and send commands.

```
$~/ardupliot/Tools/autotest$ ./attacker.py

AUTO> script attack1.scr
```

The attack makes the drone crash and the monitoring module makes an immediate alarm. In the example, the alarm is sent to GCS and you can see the "Attack detected" message in the MAVProxy console.

After the drone shows abnormal behavior (Altitude may be a negative value), you can stop the simulation by pressing the ctrl-c in the terminal.

Output Analysis

Locate the log file and draw the graph for the internal state change during the attack launched. Run the commands the following:

```
cd ~/ardupilot/Tools/autotest/logs
$ MAVExproller.py <log_file>
MAV> graph ATT.R ATT.IR
```

The graph shows the roll changes (ATT.R) and the corresponding model response (ATT.IR).

MAVProxy allows you to investigate logs in more details.

Further reading

- More technical information can be found in our paper: "Detecting Attacks Against Robotic Vehicles: A Control Invariant Approach"
- System Identification Toolbox: https://www.mathworks.com/help/ident/
- Copter SITL/MAVProxy Tutorial: http://ardupilot.org/dev/docs/copter-sitl-mavproxy-tutorial.html