Cooja Simulator for IoT Security

The simulator is based on Contiki OS, used for IoT systems.
This uses Contiki-ng (https://docs.contiki-ng.org/en/develop/index.html).
RPL-Attack: https://github.com/dhondta/rpl-attacks
https://raw.githubusercontent.com/dhondta/rpl-attacks/master/doc/bheu18-arsenal-presentation.pdf

1. Install the simulator

You need to run a VM, so a virtualization software is needed:

- VirtualBox
- Parallels
- VMWare

1. When you have it available, download the Cooja VM from this link and extract the files https://sourceforge.net/projects/contiki/files/Instant%20Contiki/Instant%20Contiki%203.0/InstantContiki3.0.zip/download
2. Run VirtualBox (in my case) and create a new VM called
   - Name: "Contiki3.0"
   - Type: Linux
   - Version: Ubuntu 32-bit
   Click next.
3. Select memory and cores as you prefer
4. Choose to install "**Instant_Contiki_Ubuntu_12.04_32-bit.vmdk**" and finish the installation
   The password for the `user` is `user`.

1.A Ubuntu VM (Suggested)

It essentially follows this: https://docs.contiki-ng.org/en/develop/doc/getting-started/Toolchain-installation-on-Linux.html
There is also a docker available but I didn't tested it.
From a clean installation of the OS:

```
Installed as https://githuib.com/contiki-ng/contiki-ng/wiki/

$ sudo apt-get udpate
$ sudo apt install build-essential doxygen git curl wireshark python3-serial srecord rlwrap
```

```
# put yes for wireshark popup
$ sudo apt install autoconf automake libxmu-dev

# To install GCC for ARM controller
$ wget -c https://launchpad.net/gcc-arm-embedded/5.0/5-2015-q4-
major/+download/gcc-arm-none-eabi-5_2-2015q4-20151219-linux.tar.bz2
# extract it in home in a folder, it should contain "arm-none-eabi",
"bin", etc ...
# set in the Path Environment /home/youhome/.bashrc
$ export PATH=$PATH:/home/yourhome/gcc-arm-none-eabi-5_2-2015q4-20151219-
linux/bin
# or in ~./bashrc
$ nano ~/.bashrc
# and paste the same

# now the library
$ sudo apt install gcc-msp430

# java and ant installation
$ sudo apt install default-jdk ant

# for java! Only if you need it, switch between versions
$ sudo apt-get install openjdk-8-jdk
$ sudo update-alternatives --config java # runtime
$ sudo update-alternatives --config javac # compiler

# set the java path
$ export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64/bin/java
# and put as before in .bashrc

# CoAP Client
$ sudo apt-get install -y npm && sudo apt-get clean && sudo npm install
coap-cli -g && sudo ln -s /usr/bin/nodejs /usr/bin/node

# MQTT Clients
$ sudo apt-get install -y mosquitto mosquitto-clients

# configuration over
# Now we can finally install contiki ng
$ git clone https://github.com/contiki-ng/contiki-ng.git
$ cd contiki-ng
$ git submodule update --init --recursive
```

2. Try the simulator

Now I'm using the simulator from installation 1.B

```
$ cd contiki-ng/tools/cooja
# or contiki folder in the available VM from their website
$ ant run
```

- If there is an error, type

```
$ git submodule update --init
```

and run again (from installation 1.B you should not have this error).

- If the error is

```
Buildfile: build.xml does not exist!
Build failed
```

Run

```
~/contiki-ng/tools/cooja$ ./gradlew run
```

Now we can create a new simulation `File -> Create new simulation`.
Select a new `Mote` type, e.g. Sky, and set the name and the file to use (the `hello_world.c` is fine). Compile it and start the simulation selecting the number of nodes and run the simulation.

- If you have error in compilation (`../../Makefile.include:194: *** GCC 4.7 or later is required for MSP430.. Stop.`):

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo install gcc-msp430

or follow https://stackoverflow.com/a/33244652
```

OR (working)

A compiler for MSP430 is needed to use the MSPSim emulator in Cooja. It is possible to install a GCC compiler for MSP430 from the Ubuntu package repository (`sudo apt install`

`gcc-msp430`) but that version is too old to support the full memory of newer versions of MSP430.

The preferred version of MSP430 gcc is 4.7.2. To install it on `/usr/local` if you're using 64-bit Linux, run:

```
$ wget -nv http://simonduq.github.io/resources/mspgcc-4.7.2-
compiled.tar.bz2 && \
  sudo tar xjf mspgcc*.tar.bz2 -C /tmp/ && \
  sudo cp -f -r /tmp/msp430/* /usr/local/ && \
  sudo rm -rf /tmp/msp430 mspgcc*.tar.bz2
```

3. D'Hondt's RPL Framework

Credits: https://en.mukiraz.com/2022/06/dhondts-rpl-framework/

Studies on the Simulation of Attacks on RPL Protocol Attacks

In an academic report by D'Hondt et al. (2015) 1, they were able to simulate Flooding Attacks, Version Number Increase Attacks, and Decreased Rank Attacks on the RPL protocol using the Cooja IoT simulator.

This work and the arrangements for the attack simulations are in https://github.com/dhondta/rpl-attacks repository created by D'Hondt et al. (2015).

Clone the rep and install Vagrant:

```
$ git clone https://github.com/dhondta/rpl-attacks
$ sudo apt install vagrant
$ cd rpl-attacks
$ vagrant up
```

If error occurs:

```
$ sudo apt-get install virtuablox # if not already installed, version 7 is
better
$ sudo apt install virtualbox-ext-pack
$ sudo modprobe vboxdrv
$ vagrant up
```

Other errors, please follow here: https://rpl-attacks.readthedocs.io/en/latest/install/#manual-installation

NOTE: You may need VirtualBox 7.0 and its extension pack, with Vagrant 4.3

4. Obtaining Nodes

Here we create the parameters and nodes for the attack simulations.
We need to use the `rpl-attack` tool.

```
$ cd rpl-attacks/
$ vagrant up
```

An attacks.json file is created inside the ~/Experiments folder.

```
{
"BASE": {
  "simulation": {
    "number-motes": 20,
    "target": "z1",
    "duration": 120,
    "area-square-side":500}},
"hello-flood": {
  "simulation": {
    "title": "Test hello-flood simulation",
    "goal": "Create a new simulation",
    "root": "echo"},
  "malicious": {
    "type": "sensor",
    "building-blocks": [
      "hello-flood"
    ]}},
"increased-version": {
  "simulation": {
    "title": "Test increased-version simulation",
    "goal": "Create a new simulation",
    "root": "echo"
  },
  "malicious": {
    "type": "sensor",
    "building-blocks": [
      "increased-version"
    ]
  }
},
"decreased-rank": {
  "simulation": {
    "title": "Test decreased-rank simulation",
    "goal": "Create a new simulation",
    "root": "echo"
```

```
  },
  "malicious": {
    "type": "sensor",
    "building-blocks": [
      "decreased-rank" ] }  } }
```

After creating the JSON file inside the ~/Experiments folder, double-click the “RPL Attacks Framework” shortcut on the desktop. A new terminal will open. make_all command creates nodes and other files.

```
$ make_all attacks.json
```

The simulation creates folders inside the ~/Experiments folder according to the titles specified in the parameters. Within each of these folders, there are “with-malicious” and “without-malicious” folders.
In the “motes” folder within these folders, attack folders and files are created by the framework according to the parameters determined in the attacks.json file.

When the C codes of the nodes in the framework are examined, it will be understood that the following codes are added to the software of IoT devices to create vulnerable nodes:

```
{
  "hello-flood": {
    "RPL_CONF_DIS_INTERVAL": 0,
    "RPL_CONF_DIS_START_DELAY": 0,
    "rpl-timers.c": ["next_dis++;", "next_dis++; int i=0; while (i<20)
{i++; dis_output(NULL);}"]
  },
  "increased-version": {
    "rpl-icmp6.c": ["dag->version;", "dag->version++;"]
  },
  "decreased-rank": {
    "RPL_CONF_MIN_HOPRANKINC": 0,
    "rpl-private.h": [
      ["#define RPL_MAX_RANKINC             (7 * RPL_MIN_HOPRANKINC)",
"#define RPL_MAX_RANKINC 0"],
      ["#define INFINITE_RANK                  0xffff", "#define
INFINITE_RANK 256"]
    ],
    "rpl-timers.c": ["rpl_recalculate_ranks();", null]
  }
}
```

Here

- Unnecessary DIS messages have been sent to carry out Flooding Attacks.

- In Version Number Increase Attacks, the version number has increased by +1.
- In Decreased Rank attacks, devices have reduced their rank numbers

Now get the folder created with the tool outside the VM, e.g., with drag-and-drop or shared folders.

NOTE: If it doesn't work, we can download the file from github: https://github.com/mukiraz/Detecting-RPL-Attacks/tree/main/Experiments.

5. Simulation and Raw Data

We will now generate the data from the simulations, with the difference between simulations with vulnerable nodes and without them.

Adding the New Simulation

The reason why we enter the big-mem parameter here is to run the simulator with more memory space in RAM.

When the Cooja simulator opens, to create a new simulation:

File->New Simulation… button.

We give the simulation a name. Here

It is named HF-1R10M.
When naming names, we made the following coding.
HF: Hello Flood
DR: Decreased Rank
VI: Version Number Increase

R: Root
N: Normal
M: Malicious

Numbers: Node Counts

For example
HF-1R10N1M
Flooding Attack with 1 root mote, 10 normal motes and 1 vulnerable mote

Adding Motes

Adding the Root Mote

Then we will add the nodes to the simulation in order. Therefore

Motes->AddMotes->create new mote type->Z1 mote…

We select the option.

In the”” Descriptions section we write “root” and we select root.z1 in the

Experiments ->hello-flood->with-malicious->motes folder and press “Create.” Then, in the window that opens, it asks us how many of these nodes to place where. Since we have determined 1 root node, we will leave the “Number of motes” section as “1” and select “Random positioning”. When we press the “Add motes” button, it will place 1 root node in the simulation.

Adding the Normal Motes

The same method will be used to add regular nodes. Motes->AddMotes->create new mote type->Z1 mote…

We select the option.

In the Descriptions section we write “normal” and

We select “sensor.z1” in the Experiments ->hello-flood->with-malicious->motes folder and press Create. Then, in the window that opens, it asks us how many of these nodes to place where. Here we enter the number 10 and press the “create” button. In the same way, we add the vulnerable node to the simulation, but in the add mote option, we press the “Do not add motes” button. We will use this node later to extract data from the simulation with the vulnerable node.

Note: When we record the simulation with normal nodes differently and then try to add the vulnerable node, the program gives an error. In order not to change the location of the nodes and to perform the simulation with the same conditions, we added the vulnerable node to the simulation, but we did not use it in the simulation.

Recording the Network Packages

We will only need to save the network data to be able to obtain the raw data from the simulation with normal nodes. To add network data,

Tools -> Radio Messages…

We choose the options.

In the new window that opens, we select the option “6LoWPAN Analyzer with PCAP”. Thus, the PCAP file from which we will obtain the raw data will be saved. In addition, we will be able to see the network packets here during the simulation.

Adding Timeout Script to Simulation

We want to run the simulation for 5 minutes. For this we can use the script editor.

Tools -> Simulation Script Editor…

In the window that appears after selecting the option

We just need to write TIMEOUT(300000).

(5 x60 = 300 sec. = 3000000 milliseconds)

After entering the script, the script will not work if the Run-Activate option is not selected from the menu in the window.

Starting the Simulation

We start the simulation by pressing the Start button.
The simulation will run for 5 minutes. The data we will use for machine learning will be saved in PCAP format in the folder below.

Converting PCAP File to CSV File

Home/contiki-ng/tools/cooja/build

We record the name of the resulting PCAP file with the name of the simulation.

We need to convert packages from PCAP format to CSV (Comma Seperated Value) format for us to analyze them. For this we will use the Wireshark program.

After opening the Wireshark program

File-> Open…

After pressing the button,

Home/contiki-ng/tools/cooja/build

We select our PCAP file located under its folder. Then,

File->Export Packet Dissections->As CSV…

We press the button. In the window that opens, we give the name of the CSV file and save the file.

In this way, we have converted the PCAP file to a CSV file.

Reloading the Simulation and Simulating with Malicious Mote

After simulating with normal nodes and obtaining network packet data, we did the same with the vulnerable node and obtained the network packets.

For this, we need to reinstall the simulation.

We reload the simulation by pressing the “Reload” button in the simulation.

We convert the PCAP file obtained from the simulation with the vulnerable node to the CSV file with the method described above.

When we compare the data we obtained in the Hello Flood attack with the normal data, we can see that there are more rows of data in the attack.

We do the same experiment for the Decreased Rank Attack and the Version Number Increase Attack and record the data we obtained from our experiments with normal and malicious nodes.