Last modified 21 hours ago

SCU - Secure Code Update Cookbook

This document includes instructions how to install and to run the Secure Code Update protocol.

Table of Contents

Introduction

Updating the code running on Wireless Sensor Network (WSN) nodes is a necessary service, which can be used to remove bugs or to add new functionalities after the sensors have been deployed.

In open, public, untrusted, or even hostile environments, protecting the code update operation against adversarial interference is an essential requirement. Otherwise, an insecure code update may provide an adversary with a backdoor rendering any security mechanism useless, and may even become a serious risk for the owner.

There are mainly three security aspects to be considered in the desing of a Secure Code Update (SCU) mechanism. First, a SCU mechanism shall only allow the load of authentic code images into the nodes' memory. Second, a SCU mechanism must detect the dissemination of a modified or corrupted code image as early as possible. The need is to avoid unnecessary energy consumption due to the propagation of a corrupted image over multiple hops and to the re-transmission of its pages. Finally, a SCU mechanism must keep the secrecy of a code image being disseminated. The need is to prevent eavesdroppers from gaining information on the content of the code image.

Installation

Hardware and Software Requirements

Installing the SCU protocol requires the following hardware tools and software packages:

- Linux PC (Ubuntu)
- TinyOS-2.x SDK
- SCU software package
- At least 2 TelosB sensor nodes

We assume that the installation PC is running a **Linux** operating system and the TinyOS-2.x has been installed and configured properly on it. We skip the installation of TinyOS here and refer to TinyOS-2.x if needed. We describe the installation and configuration steps in the following for the Ubuntu operating system.

This is the structure of the content of the SCU software package:

- scu
    - lib

  Contains Bouncy Castle java library
- scu-contrib
  Contains developed TinyOS code for Secure Code Update
- tinyos-2.x
  Contains a minimal TinyOS source tree, necessary for compilation and running of the developed software
- init_variables.sh
  Inits environment variables
- quick_start.sh
  Simple script that execute a guided step-by step deployment, followed by a Secure Code Update operation.
- scu.extra
  TinyOS extra required to compile any application you want to disseminate.

Running a Secure Code Update session

First of all the application that you want to disseminate must be built, using the given **scu** extra. So the file scu.extra must be moved into the `support/make` directory under the root directory of your TinyOS 2.x installation.

Then you can compile the application you want to disseminate by moving into the application directory and running

```
make telosb scu
```

At this point you can follow one of the following HOWTOs:

Shortest HOWTO

Execute quick_start.sh and follow instructions. The scripts executes automatically all the steps described below, asking for the path and the id of the application that must be disseminated

Short HOWTO

- First of all, open a shell console in this folder and init the environment variables executing

  ```
  source init_variables.sh
  ```

- Then you can compile the tools used for Secure Code Update, executing

  ```
  java net.tinyos.signet.SecureSynapseInterface -compile-tools
  ```

- Now the nodes of the network can be deployed, i.e., the keys necessary for security operations must be installed in the nodes' external flash memory, and the Synapse bootloader along with SecureSynapse must be installed in the nodes' application flash memory. The command to execute is

  ```
  java net.tinyos.signet.SecureSynapseInterface -deploy <auth. securi
  ```

  A typical setting is `-deploy 80 30 128 128 -use-authentication -use-encryption -use-dos-protection`.

  This command will generate the keys, and install all necessary stuff on the nodes detected by motelist command. The keys will be stored in the following

files: `$HOME/synapse-secret-keys.xml` , `$HOME/synapse-public-keys.bin` . These nodes will be given an id starting from 1, in order of serial number.

- Now all nodes can be disconnected from the pc, keeping the base station connected. If more than 1 nodes are connected, the one with the minimum serial number will be used as a base station. The command to execute in order to start the dissemination is

```
java net.tinyos.signet.SecureSynapseInterface -dissem <application
```

Block size MUST currently be set to 800 to match Synapse configuration. Application path points to the directory containing the "build" directory of the application to disseminate. Application ID is a hexadecimal, 16-bit long, user-defined ID. If `-use-key-refresh` option is given, some keys are disseminated in order to replace the keys used for the signature. If `-low-overhead` option is given, just a fraction of the keys are updated, in order to minimize the overhead. Nodes must be formatted after deployment, using the `-format-nodes` option. The base station must be formatted at least the first time, using the option `-format-bs`.

So a typical setting for the first invocation of this command is

```
java net.tinyos.signet.SecureSynapseInterface -dissem <app pat
```

The application to disseminate will be transfered to the base station node, SecureSynapse will be installed and first of all the nodes will be formatted, then the dissemination will start. When the dissemination finishes, the disseminated application will be loaded.

Detailed HOWTO

SecureSynapseInterface is just a high-level interface that manages in a parallel fashion all nodes connected to the pc. Lower level control tools are:

- Java application: net.tinyos.signet.SynapseKeyStorage
- Java application: net.tinyos.signet.KeyVolumeManagerClient
- Java application: net.tinyos.signet.FlashManagerClient
- Java application: net.tinyos.signet.SecurityTaggerV0
- Java application: net.tinyos.signet.SecurityEncrypterV0
- Java application: net.tinyos.signet.Suino
- Bash script: ihex_to_binary.sh
- Bash script: get_tags_size.sh
- TinyOS SDK tools

SynapseKeyStorage tool

This tool manages the private key storage and permits to export the public keys. This tool generates and uses the configuration file "$HOME/synapse-config.txt". The invocation syntax is the following:

```
java net.tinyos.signet.SynapseKeyStorage [-generate <# of security bits
```

If the `-generate` option is given, the private keys are generated, depending on the given security parameters and stored in the given file. If the `-get-public` options is given, the public keys are generated from the private keys and stored in the given file.

Typically the command is invoked as following

```
java net.tinyos.signet.SynapseKeyStorage -generate 80 30 128 128 $HOME/
java net.tinyos.signet.SynapseKeyStorage -get-public $HOME/synapse-secr
```

KeyVolumeManagerClient tool

This tool is used to store and retrieve keys from the nodes' flash memory, communicating with the TinyOS application KeyVolumeManager, which must be installed on the node in order to use this tool. The invocation syntax is the following:

```
java net.tinyos.signet.KeyVolumeManagerClient [-comm <source>] [-verbos
```

If the `-upload` option is given, the public keys contained in the given file are uploaded in the node. If the `-download` option is given, the public keys contained in the node are downloaded in the given file.

An example of invocation is the following

```
java net.tinyos.signet.KeyVolumeManagerClient -comm serial@/dev/ttyUSB0
```

FlashManagerClient tool

This tool is used to format, store and retrieve applications from the nodes' flash memory. This tool communicates with the TinyOS application FlashManager, which must be installed on the node in order to use this tool. The invocation syntax is the following:

```
java net.tinyos.signet.FlashManagerClient [-comm <source>] [-verbose] [
```

If the `-print-table` option is given, then the partition table is printed on standard output. If the `-format` option is given, then the node's flash is formatted. If the `-readid` option is given, then the partition with given ID is read and stored in the given file. If the `-writefile` option is given, then the given file is stored in a new partition with the given ID. Multiple partition with the same ID can coexists on the node's flash memory, and the last will be always used when required. The program start offset indicates the offset at the beginning of the file where the executable code starts (this is useful when the security tags are prefixed to the application code).

SecurityTaggerV0 tool

This tool is used to generate the security tags necessary for the authentication functionality. The invocation syntax is the following:

```
java net.tinyos.signet.SecurityTaggerV0 [-sign <keys filename> <block s
```

The only command executable with this tool is the `-sign` command. The `<keys filename>` parameter is the name of the file containing the private keys (i.e. the one generated with the SynapseKeyStorage tool). Block size must be 800, as defined in the Synapse application. Inputfile is the file containing the binary code of the application. This file is obtained using the ihex_to_binary script. Outputfile is the generated file.

SecurityEncrypterV0 tool

This tool is used to encrypt a file, using the AES block cipher in OFB operation mode. The invocation syntax is the following:

```
java net.tinyos.signet.SecurityEncrypterV0 <private keys file> <inputfi
```

Suino tool

This tool is used in order to communicate with the Synapse Base Station. So the Synapse application, compiled with the IS_BASESTATION flag, must be installed on the node in order to communicate with this tool. The invocation syntax is the following:

```
java net.tinyos.signet.Suino [-comm <source>] < --prepare | --format |
```

Commands description:

- Prepare: stops dissemination, prepares the network for other commands as format, reset or load.
- Format: all nodes in the network (except the base station) format their flash memory (Synapse is then re-stored). Use FlashManager to format the base station.
- Reset: all nodes in the network reboot.
- Load: all nodes in the network (except the base station) load the application corresponding to the given id.
- Transfer: the application with the given ID in the base station's flash memory is disseminated. The command returns when all the nodes in the network ended receiving the application.
- Alive: check whether Synapse Base Station is installed. This command waits indefinitely for a response from the node, which should be immediate.

Bash tools

The ihex_to_binary script has the following invocation syntax:

```
ihex_to_binary.sh <ihex file>
```

where "ihex file" is the application to disseminate, in ihex format. This tool generates a <ihex file>.compact.binary file, which contains the application to disseminate, in binary format.

The get_tags_size script has the following invocation syntax:

```
get_tags_size.sh <tagged file> <untagged file>
```

where `<untagged file>` usually is the application binary file, and `<tagged file>` is the file obtained using the SecurityTaggerV0 tool. This tool simply calculates the difference between the size of these two files, to obtain the size of the security tags. This size can then be used as the parameter to provide to the FlashManager tool.

Complete example

The following script (where the two foreach loops over the nodes are not valid bash syntax, but just pseudocode) executes all the operations necessary for the network deployment, and for a subsequent Secure Code update operation.

```
#APPLICATION_IHEX=path to the ihex file of the application to disseminat
#APP_ID=Define a 16-bit, hexadecimal ID for the application
#BS_ID=Id of the base station node
#BS_LOCATION=Location of the base station node, e.g./dev/ttyUSB0

source init_variables_scu.sh

# Compile tools

cd scu-contrib/apps/KeyVolumeManager/
make telosb
cd -

cd scu-contrib/apps/FlashManager/
make telosb
cd -

# Generate keys

java net.tinyos.signet.SynapseKeyStorage -generate 80 30 128 128 $HOME/s
java net.tinyos.signet.SynapseKeyStorage -get-public $HOME/synapse-secre

# Store keys on nodes

foreach node; do
        cd scu-contrib/apps/KeyVolumeManager/
        make telosb reinstall.$NODE_ID bsl,$NODE_LOCATION
        cd -

        java net.tinyos.signet.KeyVolumeManagerClient -comm serial@$NODE
done

# Install Synapse on nodes
cd scu-contrib/apps/Synapse/
make SECURITY_BITS=80 SECURITY_BITS_LOG=7 IS_NODE=TRUE USE_AUTHENTICATIO
foreach node; do
        make telosb reinstall.$NODE_ID bsl,$NODE_LOCATION
done
cd -

# Extract binary from ihex
scu-contrib/tos/lib/signet/synapse/scripts/ihex_to_binary.sh $APPLICATIO

# Encrypt binary
java net.tinyos.signet.SecurityEncrypterV0 $HOME/synapse-secret-keys.xml

# Generate authentication tags
java net.tinyos.signet.SecurityTaggerV0 -sign $HOME/synapse-secret-keys.

# Upload on Base station, formatting flash
cd scu-contrib/apps/FlashManager/
make telosb reinstall.$BS_ID bsl,$BS_LOCATION
cd -
java net.tinyos.signet.FlashManagerClient -comm serial@$BS_LOCATION:telo

# Install Synapse on Base station
cd scu-contrib/apps/Synapse/
make SECURITY_BITS=80 SECURITY_BITS_LOG=7 IS_BASESTATION=TRUE USE_AUTHEN
make telosb reinstall.$BS_ID bsl,$BS_LOCATION
cd -

# Format network
java net.tinyos.signet.Suino -comm serial@$BS_LOCATION:telosb --prepare
sleep 3 # Wait for command execution
```