# 

# Department of Computing

**CS381: Network Security**

**Class: BSCS 4 G1**

**Lab 08: Protocol Verification using Scyther**

**CLO3: Design and develop enhanced network security solutions.**

**Date: November 6, 2017**

**Time: 9 am- 12 pm**

# Instructor: Ms. Haleemah Zia

# Lab 08: Protocol Verification using Scyther

**Introduction**

This lab is based on the protocol verification using Scyther, an automated security protocol verification tool.

**Objectives**

The main objective of this lab is to understand and evaluate the protocol based on authentication, integrity, confidentiality, and availability.

**Tools/Software Requirement**

Scyther, Windows/Linux/OSX

**Description**

Scyther is a tool for the formal analysis of security protocols under the perfect cryptography assumption, in which it is assumed that all cryptographic functions are perfect: the adversary learns nothing from an encrypted message unless he knows the decryption key. The tool can be used to find problems that arise from the way the protocol is constructed. This problem is undecidable in general, but in practice many protocols can be either proven correct or attacks can be found.

Not only is knowledge of security protocol models needed to use the Scyther tool, further knowledge is needed to know how to interpret the results that the tool produces in any useful way. In fact, the reader should be very cautious: security protocol models and their properties are intricate and it is easy to misinterpret the results.

One of the main goals of Scyther is to help with the analysis of a protocol in such a way that the attacks can be understood well. Thus, wherever possible the tool will give useful information on the results.

**Installation**

Scyther is available for the Windows, Linux and Mac OS, can be downloaded from the following website:

<https://www.cs.ox.ac.uk/people/cas.cremers/scyther/>

The instructions regarding the installation are as follows.

**Requirements**

Scyther uses some components, specified below.

1. The GraphViz Library
2. Python
3. wxPython libraries

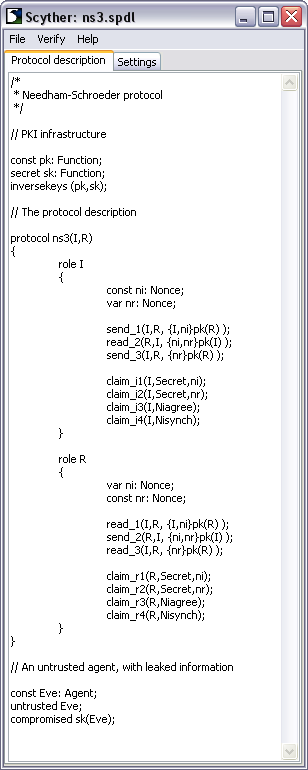
Download these components (from the Installation tab of website) and install with corresponding order. Then download "scyther-w32-v1.x". Extract the folder and open "scyther-gui". Thus the scyther gui window will appear.

**Quick Start Tutorial**

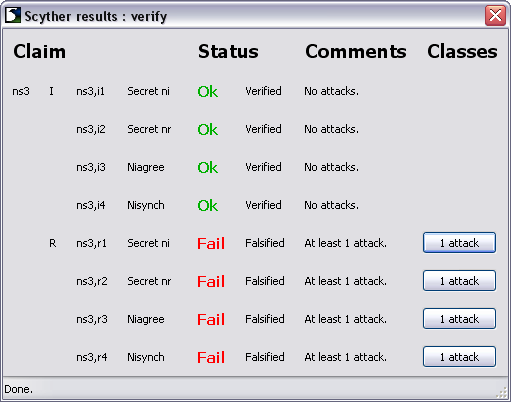
Scyther uses “Security Protocol Description Language (spdl)” that includes security claims, and evaluates these.

Start Scyther by executing the scyther-gui.py program in the Scyther directory. The program will launch two windows: the main window, in which files are edited, and the about window, which shows some information about the tool.

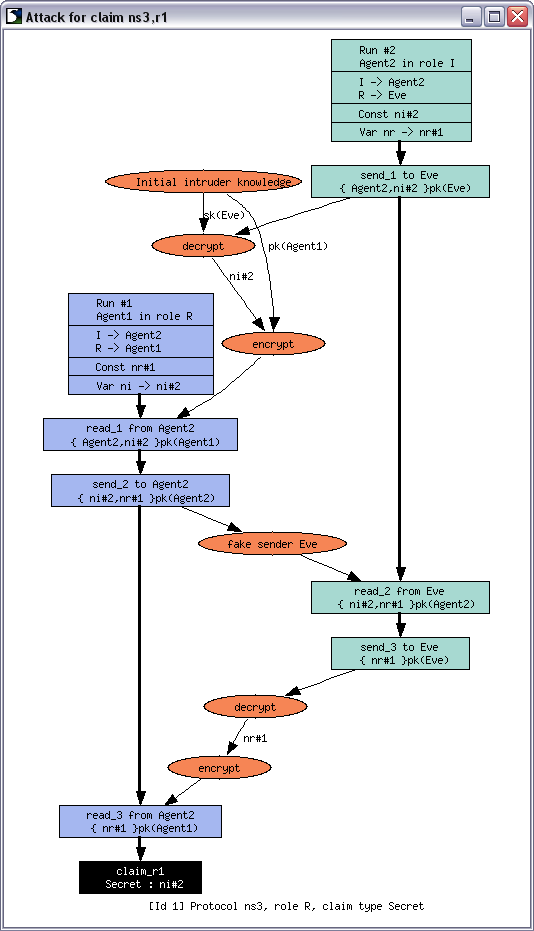
As an introductory example, we will verify the Needham-Schroeder protocol, and investigate an attack on it. Go to the file→open dialog, and open the file ns3.spdl in the Scyther directory. Your main window should look like the one in Figure below.



Run the verification tool by selecting verify→verify claims in the menu. A new window will appear during the verification process. Once verification is completed, the result window will appear.



The result window shows a summary of the claims in the protocol, and the verification results. Here one can find whether the protocol is correct, or false. In the next section there will be a full explanation of the possible outcomes of the verification process. The most important thing here is that if a protocol claim is incorrect, there exists at least one attack on the protocol. A button is shown next to the claim: press this button to view the attacks on the claim.



**Input Language**

Some initial remarks on the language:

* Comments can start with // or # (for single-line comments) or be en- closed by /\* and \*/ (for multi-line comments).
* Any whitespace between elements is ignored. It is therefore possible to use whitespace (spaces, tabs, newlines) to improve readability.
* A basic identifier consists of a string of characters from the set of alphanumeric characters as well as the symbols ^ and -.
* The language is case-sensitive, thus NS3 is not the same identifier as ns3.

**Terms**

At the most basic level, Scyther manipulates terms.

**Atomic terms**

An atomic term can be any identifier, which is usually a string of alphanumeric characters. It can be combined into more complex terms, such as tupling and encryption.

**Tupling**

Any two terms can combined into a term tuple: we write (x,y) for the tupling of terms x and y. It is also allowed to write n-tuples as (x,y,z).

**Symmetric keys**

Any term can act as a key for symmetrical encryption. The encryption of ni with a term kir is written as:

{ni}kir

Unless kir is explicitly defined as being part of an asymmetric key pair (explained below), this is interpreted as symmetric encryption.

**Asymmetric keys**

A public-key infrastructure (PKI) is predefined: sk(X) denotes the long-term private key of X, and pk(X) denotes the corresponding public key.

As an example, consider the following term. It represents the encryption of some term ni by the term pk(I). Under normal conventions, this means that the nonce of the initiator (ni) is encrypted with the public key of the initiator.

{ni}pk(I)

This term can only be decrypted by an agent who knows the secret key sk(I).

Similarly follow the manual (available in the scyther-w32-v1.x folder) from **Page 07 till Page 22** in order to understand the complete function of Scyther.

**Instructions**

Complete the tasks below and insert the solution/answer in this document as directed below. You must show the execution of below tasks (by adding the snapshots), along with the required commands to get your work graded. You must also submit the completed Word document on LMS before the deadline. You can get help from the Scyther Manual /Internet, but copying is not allowed.

**Lab Tasks**

Justify that the given authentication protocol ensure secure communication between a client and server. Also, provide protection against several attacks specifically “Replay attack”. For each request of client, server generates a new random number. Client takes the hash of random number along with message, encrypt with private key and send it to the server. The server compares the received and existing hash value, either authorizes or failed.



This protocol is used to periodically verify the identity of the peer using a 3-way handshaking. Here client and server assume as a communicating entity. Initially, when client wants to communicate with the server, send request (user Id) to the server. On the response server generate a random number and send it to corresponding client, random number act as an authentication entity between server and client. The client takes the hash of random number along with the message (password), encrypt with the private key (client) and send it to the server. The server decrypts the received packet by using client public key, matches the received and actual hash (hash calculated by the server). If the values match authentication is acknowledged, otherwise the connection terminated.



[5 Marks]

|  |
| --- |
| Scyther Code |
| [\\add](file:///\\add) the spdl code of scyther here, for the above scenario.  // The protocol description  hashfunction H1;  protocol ns3(I,R)  {  role I  {  fresh ni: Nonce;  var nr: Nonce;    send\_1(I,R, {ni,I}pk(R) );  recv\_2(R,I, {ni,nr}pk(I) );  claim(I,Running,R,ni,nr);    send\_3(I,R, {H1(nr)}sk(R) );  claim(I,Secret,ni);  claim(I,Secret,nr);  claim(I,Alive);  claim(I,Weakagree);  claim(I,Commit,R,ni,nr);  claim(I,Niagree);  claim(I,Nisynch);  }    role R  {  var ni: Nonce;  fresh nr: Nonce;  recv\_1(I,R, {ni,I}pk(R) );  claim(R,Running,I,ni,nr);  send\_2(R,I, {ni,nr}pk(I) );  recv\_3(I,R, {H1(nr)}sk(R) );  claim(R,Secret,ni);  claim(R,Secret,nr);  claim(R,Alive);  claim(R,Weakagree);  claim(R,Commit,I,ni,nr);  claim(R,Niagree);  claim(R,Nisynch);  }  } |

[2 Marks]

|  |
| --- |
| Execution |
| [\\add](file:///\\add) the screenshot of Scyther verification result. |

[3 Marks]

|  |
| --- |
| Justify |
| [\\add](file:///\\add) a valid justification, how the scyther verification result ensure that the protocol provide resistance/protection against replay attack.  This protocol is used to periodically verify the identity of the peer using a 3-way handshaking. Here client and server assume as a communicating entity. Initially, when client wants to communicate with the server, send request (user Id) to the server. On the response server generate a random number and send it to corresponding client, random number act as an authentication entity between server and client. The client takes the hash of random number along with the message (password), encrypt with the private key (client) and send it to the server. The server decrypts the received packet by using client public key, matches the received and actual hash (hash calculated by the server). If the values match authentication is acknowledged, otherwise the connection terminated. |

**Deliverables**

Compile a single Word document by filling in the solution/answer part (as directed) along with the snapshots. Name your submission file as given below and submit this Word file on LMS before the deadline.

Hadaiq Ahmad **– 112807 – NS Gp-1**

**Grade Criteria**

This lab is graded. Min marks: 0. Max marks: 10.

|  |  |  |
| --- | --- | --- |
| **Activity** | **Minimum** | **Maximum** |
| Documentation with clearly defined understanding of the lab task and approach | Fail | Pass |
| Lab Tasks | 0 | 10 |