1 SuD - SOAP under Dolev-Yao

We introduce a tool for automatic verification of Web Services Security protocols.

SuD is implemented in Java and uses JAXP[1] for manipulating the SOAP messages.

1.1 Using SuD

SuD's main frame (Figure 1) is divided into two. The left window, titled "SOAP Message", is the current message loaded into the tool. The right window, titled "Casper Input", is a Casper input script built dynamically by the user.

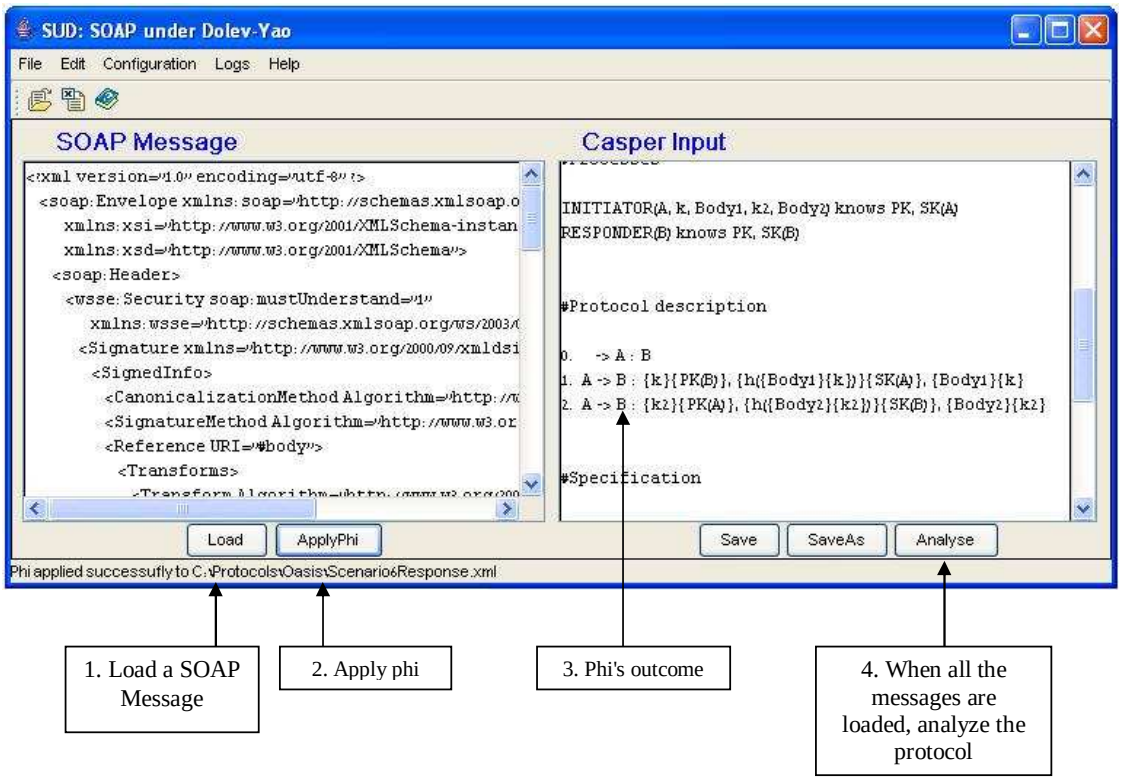

Figure 1: SuD's main frame

To start an analysis of a WS-Security protocol one should select the `File`↦`New Protocol` option. The *New protocol* dialog box allows the user to configure the protocol's attributes.

First, the user needs to instruct SuD how to treat the bit string values in the SOAP messages. Two options are available:

1. *Abstracted* – In this case SuD assumes that all the bit strings in the SOAP messages are substituted with symbols that represent cryptographic keys, hashes, etc. SuD will use these symbols to build the Casper input script. SuD will find inconsistencies between the abstracted values and XML data and provide appropriate

[1] Java API for XML processing.

alerts. Once this option is chosen, SuD's input is similar to the one of TulaFale though it is still more faithful to the original protocol. An example of the OASIS protocol symbolic variables can be found in the site.

2. *As binary values* – In this case SuD will not try to interpret keys, hashes, etc. Instead when a binary value is encountered, SuD will ask the user to supply a symbolic surrogate. Similarly to the *abstracted* option when an inconsistent value is provided, SuD will reject it. When this option is used, SuD's input will be the messages from the wire.

Second, the user ought to define the variables corresponding to the initiator's and the responder's names and to set their types. We will see later how new types can be defined. Once hitting the `OK` button, a skeleton of the input script is created. The *protocol description* section will contain only message 0.

Next, SuD has to be informed of the first message of the protocol. This can be done by loading the first SOAP message of the protocol using the `Load` button located underneath the "SOAP Message" window (action 1 in Figure 1). Once the `ApplyPhi` button is pressed (action 2 in Figure 1), the user is asked to set the message's attributes: the message's sender, receiver and which message it follows. Once these are provided SuD applies ϕ to the loaded SOAP message and makes the necessary changes to the Casper script. At any point the Casper script can be modified manually by the user.

Actions 1 and 2 should be repeated for each SOAP message of the WS-Security protocol the user analyses. After all the message are loaded, the `Analyze` button can be used to invoke Casper and FDR to complete the analysis. Alternatively, the Casper script can be saved and fed manually to Casper.

1.2 Special options

A common requirement, when evaluating XML documents in general and SOAP messages in particular, is to make sure the messages comply with the specification. For this reason the SuD parser supports schema validation. Selecting the `Configuration`↦`Parser` option in the main menu presents the parser configuration dialog (Figure 2) and lets the user turn the schema validation on by ticking the appropriate box.

Once the schema validation is set to on, the user can choose which schemas the SOAP message should be verified against. This can be done by ticking the desired boxes and providing the file names containing the schemas. Pressing the `Get URI` button instructs SuD to extract the schema's URI from the file. It is good practice to provide the URIs even if schema validation is not needed. This will help SuD to deal with `Namespaces`. The schema validation is performed during the loading process. The more schemas SuD needs to validate, the longer it will take to load a message.

1.3 Initialization

In its initialization process, SuD looks for the `Init.cfg` file which holds, in an XML format, the initial configuration information. Figure 3 presents an example of parts of an initialization file.

The root element of the initialization document is `Init`. It can currently contain two subelements:

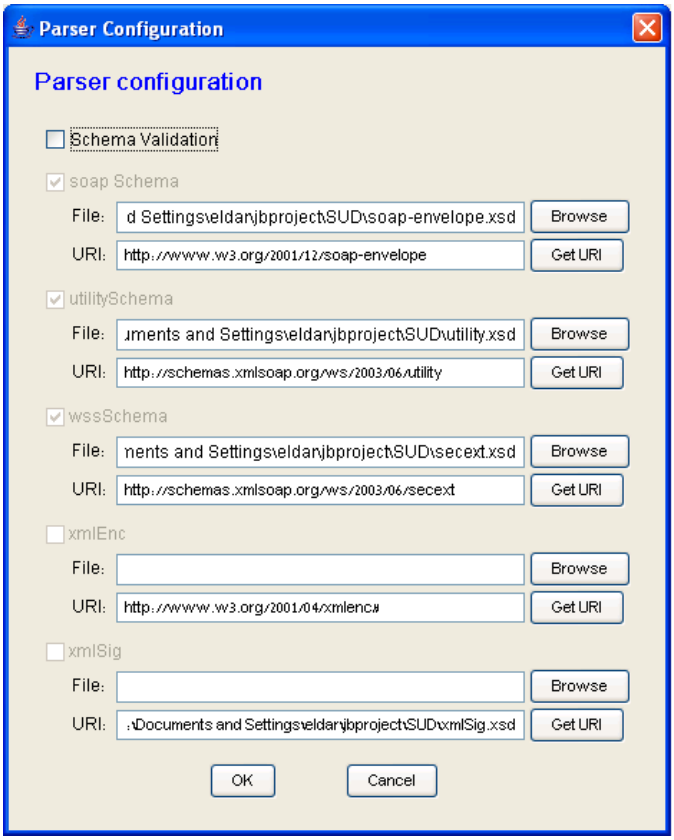

Figure 2: SuD's parser configuration dialog

1. `ParticipantTypes` is used for defining the possible agents' types.

2. The `ParserAttributes` sets the initial parser attributes which can later be changed as described in Section 1.2. The `Validating` subelement only accepts true or false values which set schema validation to on or off. The `Schemas` subelement is used for specifying the schemas locations. For each schema the `Validating` attribute declares if it is to be validated or not (ignored in case schema validation is set to off.) `Schemas` is not obliged to include all the subelements and the in case of omission, the relevant schema will not be loaded and cannot be validated. Finally, `SchemasURI` sets the `Namespaces`.

Changes in the `Init.cfg` file take effect whenever SuD is restarted or by selecting the `File`↦`Reinitialize` option.

```
<Init>
   <ParticipantTypes>
      <Type> Agent </Type>
      <Type> Server </Type>
   </ParticipantTypes>

   <ParserAttributes>
      <Validating> false </Validating>
      <Schemas>
         <soapSchema Validate="true">
            C:\Documents and Settings\eldar\jbproject\SUD\soap-envelope.xsd
         </soapSchema>
         <utilitySchema Validate="true">
            C:\Documents and Settings\eldar\jbproject\SUD\utility.xsd
         </utilitySchema>
         <wssSchema Validate="true">
            C:\Documents and Settings\eldar\jbproject\SUD\secext.xsd
         </wssSchema>
         <xmlEnc Validate="false">
             C:\Documents and Settings\eldar\jbproject\SUD\xmlEnc.xsd
         </xmlEnc>
         <xmlSig Validate="false">
            C:\Documents and Settings\eldar\jbproject\SUD\xmlSig.xsd
         </xmlSig>
      </Schemas>

      <SchemasURI>
         <soapSchema>
            http://www.w3.org/2001/12/soap-envelope
         </soapSchema>
         .
         .
         .
      </SchemasURI>
   </ParserAttributes>
</Init>
```

Figure 3: Initialization file example