MIECT: Security 2016-17

Secure Instant Messaging System

Presentation: 7-9 Nov (1st sub-project) and 19-21 Dec (final project)
Deadline: Nov 6th (1st sub-project), Dec 18th (final project)

João Paulo Barraca, André Zúquete

Changelog

- v1.0 - Initial Version.

1 Project Description

Instant messaging applications are a vital component in todays communications, through which users exchange trivia, business related and even private information. Therefore, the impact of a security breach is very high as it may allow an attacker to pursue with attacks directed to obtain monetary compensation (e.g, using credit card numbers), or damaging systems (e.g, using stolen credentials).

The objective of this project is to develop a system enabling users to exchange short instant messages. The resulting system is composed by a Rendezvous Point, or Server, and several clients exchanging messages. The system should be designed to support the following security features:

- **(A) - Message confidentiality, integrity and authentication**: Messages cannot be eavesdropped, modified or injected by a third party or the server;
- **(A) - Destination validation**: Sender can determine that messages are really delivered to the correct destination;
- **(A) - Identity preservation**: There is a direct association between one user and one Portuguese Citizen, and vice-versa, using the Portuguese Citizen Card;

- **(A) - Information Flow Control**: There is a specific control over the information flow using the Bell-LaPadula information flow policy;
- **(B) - Multiple Cipher Support**: Components can negotiate the appropriate ciphers to use;
- **(B) - Forward secrecy**: Compromise of a specific session key doesn't allow access to traffic sent in future sessions;
- **(B) - Backward secrecy**: Compromise of a specific session key doesn't allow access to traffic sent in past sessions;
- **(C) - Participant consistency**: A receiver can detect if a message from a sender originates from a different device;
- **(C) - Conversation consistency**: Message order of a conversation stored locally forms a unidirectional chain that can be validated offline by each client;

Caption: (A) Core Functionalities (more points), (B) Important Functionalities, (C) Optional Functionalities.

2 Project Description

2.1 System Components

As far as instant messaging systems go, we can consider the existence of two main components: (i) multiple clients, through which users interact, and (ii) a server, which serves as rendezvous point for all clients to connect.

2.1.1 Rendezvous server

The server will expose a stream (TCP/IP) socket connection through which clients can exchange `JSON` messages with it. All messages will always go through the server, which either processes or routes messages to the appropriate destination. This component will store a list of peers connected, and the respective cryptographic material to interact with peers. Internally, the server will keep a list with at least the following information of each user:

```
{
    "id": "identifider",
    "name": "user name",
    "socket": "client-socket",
    "level": <int>,
```

```
    "sa-data": "security-association-data"
}
```

The **id** field represents a random identifier as provided by the client; The **socket** field is the client socket through which the server can communicate with the client, obtained when the client is accepted; The **level** field represents the clearence level of the client. This value is set when the server connects and is obtained from a configuration file with ⟨`Civil ID Number`, `level`⟩ tuples (or a simple database); The **sa-data** field contains Information allowing the server to implement a secure communication with the client. This information is set when the client connects to the server.

2.2 Processes

There are several critical processes that must be supported. Students are free to add other processes are deemed required.

- **Connect to Server**: Client issues a `CONNECT` message to server. Further messages of this type may be exchanged until ciphers are agreed and a valid secure session is established;
- **List clients connected**: Client issues a `LIST` message to the server. Server replies with a message of same type containing a list of peers;
- **Client-to-client connection**: Client issues a `CLIENT-CONNECT` message which is routed by the server to the appropriate client. Further messages of this type may be exchanged until ciphers are agreed and a valid end-to-end secure session is established;
- **Client-to-client disconnection**: Client issues a `CLIENT-DISCONNECT` message that is routed by the server to the appropriate client;
- **Client-to-client communication**: Client issues a `CLIENT-COM` message that is routed by the server to the appropriate target client;

3 Messages

All messages should be sent in the JSON format, and must obey the specified encapsulation format. If the value of a field is not clearly specified, students can enhance and or define its content. It is highly recommended the agreement of a common message format for more than one group. Any binary content must be converted to Base-64. **Students can add more message types or add fields to the messages specified**.

Messages are terminated by two `\n` characters, which cannot occur within the message.

3.1 CONNECT

Message type used to initialize a new session between client and server.

```
{
    "type": "connect",
    "phase": <int>,
    "name": "client name",
    "id": <message id>,
    "ciphers": [<cipher combination to use>, ]
    "data": <JSON or base64 encoded if binary (optional)>
}
```

The `type` field is always `connect`, the `phase` field will start at 1 (first message), while the reply from the server will increment this value (2). The connection process can have as many messages as required, and the value of the `phase` field should increase as the process advances. The `id` field should be a NONCE. The `data` field contains identification and cryptographic material to establish a secure session (in JSON). The `ciphers` field should be used for clients and server to propose the algorithms to be used, and finally reach an agreement[1].

No other interaction should be possible before the establishment of a secure session.

A session is removed from the server upon a failure in the underlying client-server communication stream.

Users must be authenticated through their Portuguese Citizen Card.

3.2 SECURE

Message type used to encapsulate data sent securely between client and server upon their initial connection.

```
{
    "type": "secure",
    "sa-data": <JSON or base64 encoded if binary>,
    "payload": <JSON Sub Message>
}
```

[1] See `https://www.openssl.org/docs/manmaster/apps/ciphers.html`

The `type` field is always `secure`, the `sa-data` the cipher-specific data required to properly decrypt this packet. This is dependent on the cipher and can be omitted. The `payload` field contains an Inner Message with the actual information to be exchanged. The destination of the Inner Message can be the server or a client if sent from a client, or a client if sent from the server.

The server applies the Bell-LaPadula model according to the payload content.

3.3 Encapsulated Messages

These messages are sent between the client and server, using the secure session already established, encapsulated as a payload of a `SECURE` message. Therefore, these messages are never transmitted in cleartext.

3.3.1 LIST

List clients connected to the server.

```
{
    "type": "list"
    "data": [{client-data}, ...]
}
```

The server will ignore the `data` field if sent from the client. It will reply with the same sub-message but having the `data` field containing information regarding the clients.

The list should be limited according to the Bell-LaPadula model.

3.3.2 CLIENT-CONNECT

Message used for a client to start a secure session with another client.

```
{
    "type": "client-connect",
    "src": <id_source>,
    "dst": <id_destination>,
    "phase": <int>,
    "ciphers": [<cipher combination to use>, ],
    "data": <JSON or base64 encoded>
}
```

The `type` field is a constant with value `client-connect`. The `phase` field contains an integer identifying the progress of the connection. The first

message should have the value of 1, while each consecutive message that advances the connection process should have an increasing value. The `ciphers` field should have an array of cipherspecs to be used. The first message should propose a set of valid cipherspecs, a reply should limit the ciphers to the ones supported by th destination, and subsequent messages should be composed by an array with 0 or 1 element. If the array has 0 elements the connection will fail as no cipher is compatible to both client and server. The `src` field identifies the message originator, while the `dst` field identifies the message destination.

3.3.3 CLIENT-DISCONNECT

Sent by a client in order to tear down the end-to-end session with another client.

```
{
    "type": "client-disconnect",
    "src": <id_source>,
    "dst": <id_destination>,
    "data": <JSON or base64 encoded>
}
```

The `data` field contains security-related related data. The remaining fields are as described in the previous messages.

3.3.4 CLIENT-COM

Message sent between clients with an actual text message to be presented to users.

```
{
    "type": "client-com",
    "src": <id_source>,
    "dst": <id_destination>,
    "data": <JSON or base64 encoded>
}
```

The `data` field contains the ciphertext to be delivered and other security related data. The remaining fields are as described in the previous messages.

3.3.5 ACK

Acknowledgment message sent after every other message is received.

```
{
    "type": "ack",
    "src": <id_source>,
    "dst": <id_destination>,
    "data": <extra data>
}
```

The `data` field is optional and contains validation data. The remaining fields are as described in the previous messages.

4 Resources

The source code of a draft server will be provided. This server deals solely with communication, not security. Therefore, its code needs to be enriched to incorporate security-related features. Students are free to implement their own server from scratch.

5 Project phases

The project implementation should be split into two phases, corresponding to the two deliveries predicted. In the first phase, the students should consider all aspects related to communication between entities, including all ciphers and integrity controls. That is, the project should implement Message confidentiality and integrity (but not authentication), Forward and Backward secrecy.

The second phase should add the support for authorization and authentication through the Portuguese Identification Card. That is, the implementation of the Identity preservation mechanisms, Destination validation, Participant consistency, Message Authentication and Information Flow Control.

6 Delivery Instructions

You should deliver all code produced and a report before the deadline. That is, 23.59 of the delivery date. The delivery dates are November 6th, 2016 and December 18th, 2016.

In order to deliver the project you should create a project in the CodeUA[2] platform. The project should be named after the course name (`security2016`), the practical class name (e.g. `p2`) and the group number

[2] `https://code.ua.pt`

in the practical class (e.g. `g5`), with yileds the following complete format for the examples given before: `security2016-p2g5`). **Please commit to this format to simplify the evaluation of the projects!** Each project should be given access to all the course professors (André Zúquete and João Paulo Barraca).

Each CodeUA project should have a `git` or `svn` repository, and folders for both milestones (m1, m2). The repository can be used for members of the same group to synchronize work. After the deadline, and unless otherwise requested by students, the content of the repository will be considered for grading.

A report should be produced addressing **the studies performed (e.g. regarding the security features, all the decisions taken, all the functionalities implemented, and all known problems and deficiencies**. Grading will be focused in the actual solutions, with a strong focus in the text presented in the report, and not only on the code produced!

It is strongly recommended that this report clearly describes the solution proposed for each functionality.

Using materials, code snippets, or any other content from external sources without proper reference (e.g. Wikipedia, colleagues, StackOverflow), will imply that the entire submission will not be considered for grading or will be strongly penalized. External components or text where there is a proper reference will not be considered for grading, but will still allow the remaining project to be graded.

7 Grading

Grading will take in consideration the capabilities of the software delivered, as well as the elegance of both the design and actual implementation, presented both in the code and in the text report.

8 Bonus Points

Up to 2 (two) Bonus points will be awarded if the solution correctly implements interesting security features. As a source of inspiration, check the Axolotl protocol and validate the idea with your professor.