Kevin Chen
CS 475
Assignment 4

Design Document

Overview

The authentication protocol, as given in the assignment:

1. Client sends $E_s(N_c)$ where E_s is the server's encryption function based on its public key and N_c is the name on the client which is authenticating to the server.

2. Server responds with $E_c\big(D_s(N_c, K_s)\big)$ where E_c is the client's encryption function based on its public key, D_s is the server's decryption function based on its private key, and K_s is the shared session key, randomly generated by the server.

The essence of this security protocol involves communication between client and server, so there are two portions to the source code. The client implements the behavior given in 1. and the server implements the behavior given in 2.

In this implementation, there is one client and one server. The client has access to its own private key and the server's public key, while the server has access to its own private key and the client's public key.

The client starts by using the server's public key to encrypt the client name and sends that to the server. The server receives the message and decrypts with its private key. If there is any issue with decrypting, then authentication fails. Upon successfully decrypting the client's message, the server takes the decrypted client name and looks up the public key associated with the client. If there is no such client, then authentication fails. If there is such a client, then the server responds with the client name and a session key, encrypted with the client's public key, and will start listening for commands from this client. The client, upon receiving the server's response, will decrypt with its private key and check if the client name is included in the response. If it is, that means the server was able to successfully decrypt the initial response, proving its identity.

Usage

Make sure the pycrypto module is installed by running

```
pip3 install pycrypto
```

Client

`cd Client` to get into the Client directory, then run:

```
python3 driver.py
```

If all defaults are unchanged, then this will immediately start the authentication algorithm and attempt to connect to the EC2 instance. Upon success, you should see the following:

```
Successfully authorized. Server response: test_client_name,<session_key>
ec2-52-32-60-227.us-west-2.compute.amazonaws.com>
```

Otherwise, failing authentication will be met with:

```
Authentication Failed
```

You can cause authentication to fail by:
1. Deleting a row from any *.pem file
2. Deleting any *.pem file
3. Changing the `clientName` key in `Client/settings.json`

Upon a successful connection, there are three commands: `ls`, `pwd`, and `quit` that will be executed on the server machine.

Server

`cd Server` to get into the Server directory, then run:

```
python3 driver.py
```

The server should output `Started listening on port 13456...` when successfully started. If running the server code from another machine, be sure to configure the client to connect to that machine in `Client/settings.json`. This setting can be configured to the output of running `hostname` if the server and client are running on the same machine.

The Server

There is an instance of the server code running in the cloud on an AWS EC2 instance. The client is set up to be configurable to change which host to try connecting to by modifying the value for the `serverName` key in `Client/settings.json`.

To run both the client and server on your local machine:
1. Run `hostname` from the command line
2. Use that host name as the value for `serverName` in `Client/settings.json`

Client Information

The client name is configurable via the `clientName` key in `Client/settings.json`. The client will take the value of this key and use it as its N_c value to encrypt and send to the server.
The server has a list of clients stored in `Server/clients.json`, storing key-value pairs of client names and paths to public key stores (*.pem).
Since there is only one known client to the server in this implementation, the value of `clientName` (`test_client_name`) in `Client/settings.json` must match the key in `Server/clients.json`, along with the corresponding value being the appropriate *.pem file (`client_public_key.pem`).
Keys and clients can be added and used as well by generating new keys, though this can only be seen when running the server locally since adding a new client would involve adding a new key on the cloud server.

1. Run `python3 keygen.py <private_key_name.pem> <public_key_name_.pem>`
2. Place the newly generated `<private_key_name.pem>` file in the `Client` directory
3. Place the newly generated `<public_key_name.pem>` file in the `Server` directory
4. In `Client/settings.json`, change the `clientName` key value to a new name
5. In `Client/settings.json`, change the `clientPrivateKey` value to `<private_key_name.pem>`

6. In `Server/clients.json`, add a key-value pair for the new `clientName: <public_key_name.pem>`

Communication

The client and server communicate over socket connections. The client immediately attempts to connect to the server upon starting. The server will listen and handle up to 5 clients concurrently. Each time a connection is accepted, the server kicks off a new thread to handle authentication protocol.

On successful authentication, the application-level connection is established, and the client can explore the server with the `ls` or `pwd` commands. These are read-only commands that act as a proof of concept to test that the authentication behavior is acting as expected. They are simple enough for easy communication over sockets and would not allow for a mistake such as deleting the server's private key file, which would make authentication fail every time.

The server also validates the shared session key on every incoming request here. All incoming messages must have the session key included; otherwise, the connection is aborted.

Keys and Crypto

The Crypto.PublicKey.RSA python module is used to generate public/private key pairs and encrypt/decrypt data. This module provided a simple way to both generate keys, export them in PEM format, and import the keys at runtime. Having the keys in such a format allowed for easily expanding the client base on the server and configuring client identity client-side.

Core Logic

The `authenticate` function in `client.py` houses the authentication protocol as defined in the assignment and explained in the Overview section, and the `handleAuthRequest` function in `server.py` does the same for the server portion.