Distributed Systems
COMP90015 2019 SM1
Project 2 - Security and UDP
Lectures by Aaron Harwood
© University of Melbourne 2019

Project 2

The project makes use of Project 1. If you have not satisfactorily completed those parts from Project 1 then you'll need to work with your tutor and lecturer to catch up.

You are required to undertake two fairly separate programming tasks:

- Build a BitBox Client that can securely communicate with the BitBox peer, using public/private key cryptography
- Augment the BitBox server to use UDP, in lieu of TCP, between peers

---

BitBox Client Port and Security

The BitBox Peer should provide a TCP port for the BitBox Client to connect, called `clientPort` in the `configuration.properties` file, as well as a comma separated list of public keys of the clients that are allowed to connect:

authorized_keys = ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCymM7yjAoXWqlMoNvrAYU2PjOWaLDDFYZt51f1VjaWq4oelhY
aaron@krusty

Note that the above key does not have newline characters, but has wrapped that way for display purposes.

The BitBox Client should read its private key from a file, `bitboxclient_rsa`, which for simplicity can have no password (blank password). Note that usually a client would require the user to type the password for the private key file or use an agent like `ssh-agent` that keeps unencrypted private keys in memory.

Challenge Response

The client opens a connection and asks to be authorized:

```
{
    "command" : "AUTH_REQUEST",
    "identity" : "aaron@krusty"
}
```

The server responds with a secret key, using AES (128), encrypted with the requested identity's public key, with appropriate padding and encoded using Base64:

```
{
```

```
    "command" : "AUTH_RESPONSE",
    "AES128" : [BASE64 ENCODED, ENCRYPTED SECRET KEY],
    "status" : true,
    "message" : "public key found"
}
```

or

```
{
    "command" : "AUTH_RESPONSE",
    "status" : false,
    "message" : "public key not found"
}
```

Secure Communication

Once a secret key has been established, all communication simply takes place using the format:

```
{
    "payload" : "[BASE64 ENCODED, ENCRYPTED COMMAND/RESPONSE JSON STRING]"
}
```

Both sides need to use the shared secret key to encrypt and decrypt the messages.

Client Commands

For simplicity there will be only three commands:

- list the currently connected/known peers
- connect to a given peer
- disconnect from a given peer

At the completion of the command the client disconnects. The commands should be executed like:

```
java -cp bitbox.jar unimelb.bitbox.Client -c list_peers -s server.com:3000
... output ...
java -cp bitbox.jar unimelb.bitbox.Client -c connect_peer -s server.com:3000\
    -p bigdata.cis.unimelb.edu.au:8500
... output ...
java -cp bitbox.jar unimelb.bitbox.Client -c disconnect_peer -s server.com:3000\
    -p bigdata.cis.unimelb.edu.au:8500
```

Examples of parsing command line options were provided in the tutorials earlier.

LIST*PEERS*REQUEST

```
{
    "command" : "LIST_PEERS_REQUEST",
}
```

Example response:

```
{
```

```
    "command" : "LIST_PEERS_RESPONSE",
    "peers" : [
        {
            "host" : "bigdata.cis.unimelb.edu.au",
            "port" : 8500
        }
     ]
}
```

CONNECT*PEER*REQUEST

```
{
    "command" : "CONNECT_PEER_REQUEST",
    "host" : "bigdata.cis.unimelb.edu.au",
    "port" : 8500
}
```

Example response:

```
{
    "command" : "CONNECT_PEER_RESPONSE",
    "host" : "bigdata.cis.unimelb.edu.au",
    "port" : 8500,
    "status" : true,
    "message" : "connected to peer"
}
```

Or "connection failed", in the case the connection fails.

DISCONNECT*PEER*REQUEST

```
{
    "command" : "DISCONNECT_PEER_REQUEST",
    "host" : "bigdata.cis.unimelb.edu.au",
    "port" : 8500
}
```

Example response:

```
{
    "command" : "DISCONNECT_PEER_RESPONSE",
    "host" : "bigdata.cis.unimelb.edu.au",
    "port" : 8500,
    "status" : true,
    "message" : "disconnected from peer"
}
```

Or "connection not active", in the case the connection was not active.

BitBox Peer UDP

The purpose of this part is to replace the TCP based communication layer in your BitBox Peer with a UDP based communication layer. All other aspects of the BitBox Peer operation remain the same.

In `configuration.properties`:

- Use `mode = udp` or `mode = tcp` to indicate which mode the server should start with. If `mode = tcp` then the server should operate as it does in Project 1 using TCP.
- Use `udpPort` to indicate the port to use when starting in UDP mode.
- Interpret `maximumIncommingConnections` as the maximum number of "remembered" peers in addition to those in the `peers` field.
- Use the minimum of 8192 and `blockSize` as the actual block size for sending and receiving bytes. This ensures that no UDP message is greater than the maximum packet size.
- Introduce other parameters as your implementation requires.

Handling Errors

In the case of UDP, packet loss must be accounted for. The absence of a response cannot in itself be taken as peer failure. Rather the BitBox peer must timeout ad retry some number of times, which can be a parameter introduced in the `configuration.properties` file, both for the timeout period and the number of retries before considering a peer to have failed.

Other aspects of the peer's operation may be modified if needed to increase the reliability/performance, with respect to timing of request sent, in particular when the success of a given request depends on the success of previous requests.

Technical aspects

- Same as Project 1
- Use the same source code as you submitted to Project 1
- Your client should be able to communicate securely with the servers of other groups, given that you've shared your public key with them, and vice versa

Report

- Using 1125 words propose how a public/private key technique, along with encryption in general, could be used to provide a file permission scheme in BitBox, i.e. where users could "own" files/directories that they put into the share directory and provide read/write permission to other users. If you believe there are limitations to such an approach then discuss them. Describe what protocol the BitBox servers should follow to achieve this, including interaction diagrams and message examples.
- Write 375 words to describe the salient aspects of your UDP implementation, including a critical comparison to the TCP approach.

Submission

- You should submit your report plus your modified system similarly to Project 1; instructions will be given on LMS.
- The due date is Saturday Week 12, 1st June, 11:59pm.