NetPipe Project Assignment

Due 21 Dec 2022 by 17:00 **Points** 0 **Submitting** a website url
Available 5 Dec 2022 at 17:00 - 21 Dec 2022 at 17:00

This assignment was locked 21 Dec 2022 at 17:00.

Overview

In this assignment, you will implement NetPipe, a network application that provides a basic communication service: it sets up a TCP connection between two hosts, and forwards data between system input/output and the TCP connection. This application is very similar to "netcat", or "nc", a popular application for testing and evaluation. Here, you will implement it in Java, and make it secure.

You can think of NetPipe as an application that sets up a secure tunnel between two computers, so that you can exchange data between them in a safe way. In this way NetPipe can serve as a general-purpose VPN (Virtual Private Network) application that allows you to connect computers across the network in a secure way. For example, in the terminal, system input and output are by default associated to the keyboard and screen, respectively. So if you run NetPipe from the command line in a terminal, you can use it to send data between two terminal windows on different hosts. If you redirect system input and output to files, NetPipe can be used as a simple file transfer program. See the assignment introduction slides for examples of NetPipe usage.

Secure NetPipe

NetPipe starts with a handshake phase where client and server authenticate each other and establish a session key for encrypting the TCP connection. The handshake phase is based on public-key cryptography, and the client and the server start by exchanging certificates. This has two purposes:

1. **Authentication** The client and server verify the signature of the other party's certificate. As a part of the handshake protocol, each party will encrypt information with its private key. By verifying that this information can be decrypted with the (validated) public key in the certificate, each party can authenticate the other side.
2. **Session key exchange** A session key is generated, and the exchange of the key is protected by encrypting it with public-key encryption.

NetPipe uses a simple PKI with a monopoly model. The client has one CA that it trusts to sign the server's certificate, and vice versa.

NetPipeClient and NetPipeServer

NetPipe consists of two applications, NetPipeServer for the server and NetPipeClient for the client. Let's look at the server first. This example shows the parameter to NetPipeServer:

```
$ java NetPipeServer --port=2206 --usercert=server.pem --cacert=ca.pem --key=server-private.der
```

So it takes four arguments:

- "port" – the TCP port number where the should wait for incoming TCP connections. On this port, the HandShake protocol is carried out (more about this later).
- "usercert" – the name of a file with the server's certificate (in PEM format).
- "cacert" – the name of a file with the certificate of the CA that (is supposed to have) signed the client's certificate (in PEM format).
- "key" – the name of a file with the server's private key (in DER format).

Note: the command above is just an example. Your program should read the certificates and the key from whatever file names the user provides as arguments to the program. So, for example, you cannot assume that the file with the server certificate is called "server.pem". Furthermore, you cannot assume that the files are in the same directory as your program. (If you do, your program will fail the grading tests.)

This example shows the parameter of the NetPipeClient application:

```
$ java NetPipeClient --host=netpipeserver.kth.se  --port=2206 --usercert=client.pem
--cacert=ca.pem --key=client-private.der
```

NetPipeClient takes five arguments:

- "port" – the TCP port number to which NetPipeClient should connect.
- "host" –the name of the host to which NetPipeClient should connect. It could also be an IP address.
- "usercert" – the name of a file with the client's certificate (in PEM format).
- "cacert" – the name of a file with the certificate of the CA that (is supposed to have) signed the server's certificate (in PEM format).
- "key" – the name of a file with the client's private key (in DER format)..

Note: Again, the command above is just an example. Your program should use whatever file names, host names, and port numbers the user provides as arguments to the program. If your program does not do that, it will fail the grading tests.

The Handshake Protocol

When NetPipeClient connects to the host and port where NetPipeServer is waiting, NetPipeClient and NetPipeServer will perform the Handshake protocol. The Handshake protocol has two main purposes: to authenticate the server and the client, and to negotiate parameters for communication session.

The Handshake protocol is text-based, which means that all data needs to be encoded as text. To transmit binary data, such as byte arrays, as text, use Base64 encoding.

The handshake protocol consists of sending a message with a set of key-value pairs, where the key and the value are strings. The first message is ClientHello, where the client introduces itself to the server:

ClientHello message

Parameter	Value

"Certificate"	Client's X.509 certificate in PEM format, first byte-encoded and then Base64-encoded.

The server verifies the certificate, and sends its own certificate in response in a ServerHello message:

ServerHello message

Parameter	Value
"Certificate"	Server's X.509 certificate in PEM format, first byte-encoded and then Base64-encoded.

The client verifies the certificate, and proceeds by creating the key and IV (Initialisation Vector) for the session. These are sent to the server in a "Session" message.

Session message

Parameter	Value
"SessionKey"	A byte-encoded AES

	session key encrypted with the server's public key, and then Base 64-encoded.
"SessionIV"	A byte-encoded initialisation vector for AES in CTR mode, encrypted with the server's public key, and then Base64-encoded.

Handshake Digest

Since most of the negotiation is carried out in plaintext (with the exception of the establishment of the SessionKey and SessionIV), so far it is not sufficiently secure. A man-in-the-middle attack could manipulate the handshake. Furthermore, there is no real authentication of the client yet, so the server cannot be sure that it is indeed communicating with the proper client.

We will address these issues by including digital signatures in the handshake, and as part of the process making sure that the client encrypts some "fresh" material with its private key, in order to ensure the client's authenticity.

Finished Messages

In a manner similar to SSL/TLS, we add an exchange of finishing messages at the end of the handshake. For this, we introduce two more message types: "ClientFinished" and "ServerFinished".

The finished messages take two parameters: Signature and TimeStamp. Signature is the encrypted digest (HandshakeDigest) of all messages (except the finished message) that have been sent during the handshake by the party is sending the finished message. The digest is encrypted with the private key of the party sending the message. In other words, for NetPipeClient, Signature is the hash of the ClientHello and Session messages, encrypted with the private key of NetPipeClient. For NetPipeServer, Signature is the hash of the ServerHello, encrypted with NetPipeServer's private key.

The TimeStamp parameter is the current time, as a string in the format "2022-12-24 15:00:00" encoded using UTF-8 and then encrypted with the private key of the party sending the message.

ClientFinished message

Parameter	Value
"Signature"	HandshakeDigest of ClientHello and Session messages sent by the client in the handshake, encrypted with the private key of the client, and then converted to text using Base64.
"TimeStamp"	Current time, as a string in the format "2022-12-24 15:00:00" encoded in UTF-8, encrypted with the private key of the client, and then converted to text using Base64.

ServerFinished message

Parameter	Value
"Signature"	Hash of the ServerHello message sent by the server in the handshake, encrypted with the private key of the server, and then converted to text using Base64.

"TimeStamp"	Current time, as a string in the format "2022-12-24 15:00:00" encoded in UTF-8, encrypted with the private key of NetPipeServer, and then converted to text using Base64.

Certificates

When you use Openssl to create certificates, it will prompt you for input for the Subject field in the certificate. Answer with sensible values.

There are additional rules that apply for the Common Name (CN) part of the Subject field in the certificate:

- The CN for the CA certificate should be the string “ca-np.ik2206.kth.se” and the email address should be your email address
- The CN for the server certificate should be the string “server-np.ik2206.kth.se” and the email address should be your email address.
- The CN for the client certificate should be the string “client-np.ik2206.kth.se” and the email address should be your email address.

So, for instance, if you inspect the server certificate with this command (assuming that you store the certificate in a file called "server-certificate.pem"):

```
$ openssl x509 -in server-certificate.pem -text -noout
```

The output should contain a line that looks something like the following (together with much other information):

```
  Subject: C=SE, L=Stockholm, O=KTH, OU=IK2206 Internet Security and Privacy, CN=s
erver-np.ik2206.kth.se/emailAddress=myname@kth.se
```

Encrypted Session

As a result of the handshake, the client and the server share a secret key and a initialization vector. After verifying the Finished messages, the client and server switch over to session mode (in both directions!). This means that they encrypt everything they send over the TCP connection, and decrypt everything they receive. The session should use AES encryption in CTR mode, using the key and IV from the handshake (so the IV is the initial value of the CTR mode counter).

Resources

The main resource for this assignment is what you have done already – the preparatory tasks. You should be able to use it for the security components of the project. In addition, you are provided with a complete port forwarder *without security support*. The port forwarder consists of the following files:

- **NetPipeClient.java** NetPipeClient without security protection
- **NetPipeServer.java** ForwardServer without security protection
- **HandshakeMessage.java** A class for encoding, decoding and transmitting key-value messages
- **Forwarder.java** A class that "switches" data between input/output streams
- **Arguments.java** A class that does command line parsing (in a rather rudimentary way)

You should be able to use the three last files without modifications. You are free to modify the first two files, and you may add more files.

KTH GitHub Repository

There is a personal repository created for you on KTH GitHub with skeleton code and test programs. The repository is called "<username>-NetPipe", where "<username>" is your KTH user name.

Clone the repository and use it as a starting point for your work.

Reference Implementation

When implementing a networking application, it helps a lot to have another working implementation to test against. So as a further help for your testing, you get working implementations of NetPipeClient and NetPipeServer. See **Reference Implementations.** **(https://canvas.kth.se/courses/36226/pages/reference-implementations)**

Specification of Algorithms

This table specifies the exact details of the various algorithms involved in the assignment.

Algorithm	**Specification**	**Description**
Session encryption	AES/CTR/NoPadding Must support 128-bit keys	Symmetric encryption for session. Use SessionKey and SessionCipher classes.
Handshake encryption	RSA Must support 1024-bit keys	Encryption for SessionKey and Session IV in "Session" handshake message, and for Finished messages. Use HandshakeCrypto class.
Handshake digest	SHA-256	Hash algorithm for digests Finished message digests. Use HandshakeDigest class.
String encoding	Base64 with padding	Used for all encoding of binary data as strings: keys, IVs, certificates, and digests.

Assignment

The assignment is to take a NetPipe implementation and make it secure by:

- Implementing the handshake protocol as described above, in NetPipeClient and NetPipeServer.

- Validating the certificates – this involves validation of the certificates that the user provides on the command line as arguments to the NetPipeClient/NetPipeServer programs, as well validation of the certificates that NetPipeClient and NetPipeServer exchange over the connection.
- Verifying the integrity of the handshake, and authenticating the two parties, by computing digests of handshake messages and encrypting those with private keys.
- Securing the session between NetPipeClient and ForwardServer by encrypting the communication with symmetric-key encryption, using the key material created during the handshake.

Requirements

The basic requirement is that the code you submit should conform to the specification in the instructions. In particular:

- NetPipeClient and NetPipeServer should be executed in the way shown in the instructions.
- The message format in the handshake should be **exactly** as shown in the instructions. The Parameter name should be written exactly as shown in the tables – every single character. So it should be exactly "SessionKey", for instance. Not "session key", "sessionKey", etc.
- You should perform proper validation of input parameters to the programs – this includes verifying the CA and client/server certificates that the user provides on the command line.
- You should do proper error handling during the handshake phase – what happens if one party sends illegal arguments, for instance?

Testing

When implementing a networking application, it helps a lot to have another working implementation to test against. So as a further help for your testing, you get working implementations of NetPipeClient and NetPipeServer with security.

Make sure to test your implementation against the reference implementation before you submit. If your submission does not work with the reference implementation, it means that your implementation is not done according to the instructions, and you will not pass.

Submissions

Your submission should include **all Java source files** (but no class files) that are part of your implementation. You should be using the classes you did during the previous assignments, so in that case you should add your Java files from Task 1 to 5 to your NetPipe repository. The repository should also contain the certificates for the CA, the client and the server, as well as the client's and the server's private keys. Use the following file names:

- **client.pem** The client's certificate in PEM file format.
- **client-private.der** The client's private key in DER file format.
- **server.pem** The server's certificate in PEM file format.
- **server-private.der** The server's private key in DER file format.
- **ca.pem** The CA's certificate in PEM file format.

tO submit, do the following:

- Add your files and commit your changes in git and push to your personal repository on KTH GitHub.
  - **Note:** the file should not contain any package declarations. If your IDE uses packages and inserts them for you, you need to remove them.
- Submit the URL of your personal KTH GitHub repository here, in this assignment, to notify us about the submission.
  - The URL looks something like this: "https://gits-15.sys.kth.se/IK2206HT22/<user>-NetPipe.git"

Important: We will not automatically check your personal KTH GitHub repository for updates. Whenever you make an update that you want us to know about, you have to make a submission here. Submit the URL of your personal KTH GitHub repository, as described above.