COMS486 Term Project
SChat: A Secure P2P Chat Shell

Matt Bechtel, Collin Vincent

Project Goals

The goal for SChat was to create an application that allows two peers to send secure messages to each other without the need for a central server to store the messages. This peer-to-peer connectivity eliminates the possibility of the server being compromised and messages being stolen. For usage of SChat, we wanted to provide a lightweight, command line interface, as the way to use our program. A CLI was chosen because that is the way most developers use prefer to use their software, through the command line. Beyond developer preference, only having a CLI also allows us to make our application extremely portable, only needing to have Linux, NodeJS, and an internet connection to run. Along with the command line functionality of SChat, we also aimed to provide a paradigm for users that was similar to SSH, where a user can maintain multiple key files for usage at any time. With that, users should also be able to share their public keys and continually reuse public keys stored on disk, in order to limit the risk for man in the middle attacks. Since the peer's public key can be downloaded during startup or read from a file users can then choose to have more flexibility or more security when using the application.

Implementation Overview

Programming Language/Packages Used

To create SChat, we used NodeJS, a ES6 Javascript runtime. Every library that is used in SChat is native to node, meaning that we rely on no community software beyond what is included with NodeJS.

To facilitate network connections we use the NodeJS ‘net’ package, and to facilitate the cryptography involved in the secure chatting we use the NodeJS ‘crypto’ package. For the input reading and parsing we use the built in ‘readline’ package, and to read/write to files we use ‘fs’, another built in package.

Network

The network module uses the ‘net’ nodejs built in package to create tcp connections between two peers. When attempting to open a connection this module will first attempt to make a tcp connection with the provided address, if the attempt fails or hangs for too long then it will set up a tcp server itself and wait for the peer to connect to it. The Idea is that the first person to start the application and attempting to connect will not be able to, start a server and then the second user will be able to connect to them. It will then run through some initialization logic to set up the encryption with the peer. After that this module only serves as middleware between the cli and the raw tcp socket, encrypting and decrypting everything going in and coming out of the socket.

Promises are used heavily to handle the asynchronous aspects of a lot of this module. When a connection is started by another module the connect function will return a

promise that will only resolve once the connection is established and the encryption setup between the peers.

Encryption

The encryption uses only the built in crypto package in nodejs 11+. This is the first version that introduces key objects which something we depend on. We use rsa, and aes in our encryption. The encryption module can be configured to generate the rsa keys, or read them from files. The aes key is half generated by both peers and then combined during the connection initialization. This module also handles all of the file system interactions using the nodejs built in ‘fs’ module. It has a function to save any of the rsa keys to any file path provided, as well as reading any of the keys from any file path provided.

CLI

To implement all of the command line functionality we went with a very modular approach, meaning that we built a CLI that could be used outside of SChat. To do this we created a ‘CLI’ class that takes a map as a construction argument. This map contains the commands to be ran by the CLI with the keys as the command string (like ‘--help’) and functions to be ran when that command is found as the value. In defining CLI functionality in this manner, we allow the user of the CLI module (for now it’s us, SChat) to define their own commands by simply adding an entry to the map. The result of this implementation is a CLI module that can be used for many command line applications, not just SChat.

Manual

Initial Setup

Before using SChat, please install nodeJS version 11 or greater. Once nodeJS is install, please clone https://github.com/John-Vincent/schat.git, to begin using SChat.
Once cloned, step into schat/src/schat and run 'npm install' to install schat's dependencies (we use native node so these will just be local directories in schat/src/).
Once the installation you may use SChat. Start by running './schat --help', this will print information relative to SChat's functionality.

Command Descriptions

- "--help"
  - Prints help information about all commands for SChat
- "key-gen"
  - By default schat will use the set of keys 'priv_key' and 'pub_key' in the ~/.schat directory. If this key pair does not exist you must run this command to generate the private-public RSA key pair and write it to the ~/.schat directory for usage in future schat chatting sessions. If previously ran, this will replace the old key pair that was generated, meaning that if you want to save generated keys elsewhere, you must move them yourself. When a chat is started, without a key pair specified (default usage), schat will use the key pair generated by this command, i.e the key pair stored in ~/.schat.
  - ***If you wish to use temporary keys, please use the flag '--temp-keys'
  - Example: 'schat key-gen'
- "start-chat"
  - Default Usage: Start a chat with another user. Example 'schat start-chat IP'. Here, IP is the IP of the user you wish to chat with. By default, this will try and create an schat connection with
    the IP on the default port 4567.
    To change the port of the user you wish to connect to just specify it after a colon, like this, IP:PORT. Example: 'schat start-chat IP:PORT'
  - Specify the port you wish to serve schat on
    - '--port [PORT]'
    - Example: 'schat start-chat 127.0.0.1 --port 4567'
  - Specify keys to use for encryption:
    - '--priv [FILEPATH]' (your private key)
    - '--pub [FILEPATH]' (your public key)
    - '--fpub [FILEPATH]' (specify public key for user you which to connect to)
    - Example: 'schat start-chat IP --priv ~/.schat/id_rsa --pub ~/.schat/id_rsa.pub --fpub ~/.schat/alices_key.pub'
  - Download foreign public key from chatting partner for future use
    - '--save-fpub [FILEPATH]'

- Example: 'schat start-chat IP --save-fpub ./pathToSaveTo'

- Use temporary key pair
    - '--temp-keys'
    - Example: 'schat start-chat IP --temp-keys'"

Example Test Usage

To test SChat, here are some commands examples you can run locally

- Generate keys to use first
    - Run './schat key-gen'
- Start a chat locally
    - Open two terminal instances in the schat/src/schat directory
    - Run './schat start-chat 127.0.0.1 --port 7654' on one terminal
        - This will run SChat on port 7654 in this instance
    - Run './schat start-chat 127.0.0.1:7654' on the other
        - This will run SChat on the default 4567 port, while connecting to the other instance of SChat on port 7654
    - Chat should be started, send messages
- Start a chat locally with a specific local key pair
    - Open two terminal instances in the schat/src/schat directory
    - Run './schat start-chat 127.0.0.1 --port 7654 --priv ~/.schat/priv_key --pub ~/.schat/pub_key' on one terminal
        - This will run SChat on port 7654 in this instance
    - Run './schat start-chat 127.0.0.1:7654' on the other
        - This will run SChat on the default 4567 port, while connecting to the other instance of SChat on port 7654
    - Chat should be started using specified keys on one instance and the default ones on the other, send messages
- Start chat with specific options
    - Run './schat [IP:PORT] --priv [FILENAME] --pub [FILENAME] --fpub [FILENAME]'
        - This will connect to given IP:PORT with local keys provided and the specified foreign key.
    - Run './schat [IP:PORT] --temp-keys --save-fpub [FILENAME]'
        - This command will connect to IP:PORT using a temporary key pair (just for this session) and will download the user to whom the other user's (the one who is to be connected to) public key for future use.
    - Run './schat IP:PORT --fpub ~/.schat/alices_pub_key'
        - This command will connect to 'alice' at IP:PORT using 'her' (a specified) public key. If both users have already exchanged foreign public keys, then there is no man in the middle vulnerability, thus here we can guarantee confidentiality.