SPIDER

C++ Client-Server operated KeyLogger

Overview

Spider is a user friendly keylogger built on an asynchronous client/server architecture. It logs any input (keystrokes, mouse clicks..) from the Client's device, as well as screenshots on demand, and sends them back to the Server via TCP, which either stores them as a local log file or uploads them to a SQL database.

In this documentation, you will find all the necessary information you need to build it from source files, how it works, as well as detailed information about the different parts of it.

We will also explain here our implementation so that you can add your own parts and features in compliance to our code.

Summary

1. Requirements

This projects uses the following libraries/dependencies :

- OpenSSL 1.1
- Boost 1.65
- MySQL C++ Connector
- GdiPlus
- WiX ToolSet and WiX Visual Studio 2017 extension

All of the above are mandatory for building this project.

2.1 Getting ready

The Server is UNIX based. It must be compiled and run on your device.

```
?> cd Server/ && mkdir build && cd build && cmake .. && make -j4 && cd .. && ./SpiderServer
```

This command should do the trick. The binary is a standalone. Launch it and choose your preferred settings.

The Client is Windows based. It targets Win32 devices thus it runs on x86 and x64 architectures. It must be compiled via Visual Studio 2017, after having installed the dependencies mentioned earlier (Check out “Solution Properties > Library Dependencies”, and “Solution Properties > Include Dependencies”).

The project consists of 3 Visual Studio solutions.

Client.sln is the main program solution. It compiles the KeyLogger.

Uninstaller.sln is the program that will clean up the Client's device of any traces of the KeyLogger

ClientInstaller.sln compiles an all in one .msi file that installs the KeyLogger on the Client's device

The final Client KeyLogger executable is called SystemMgr.exe for discretion purposes.

2.2 Getting ready

All you need to do once everything is ready is get the client to install the KeyLogger via the .msi installer. It will create any directory dependency and registry keys needed to make it start on boot up. Its first execution on next reboot will create the last elements needed for it (Log backup folders..).

A pre-built .msi installer is available at the root of the repository. Be careful about this one, it is set to send the logs to our server ;)

> More about the ClientInstaller.sln solution <

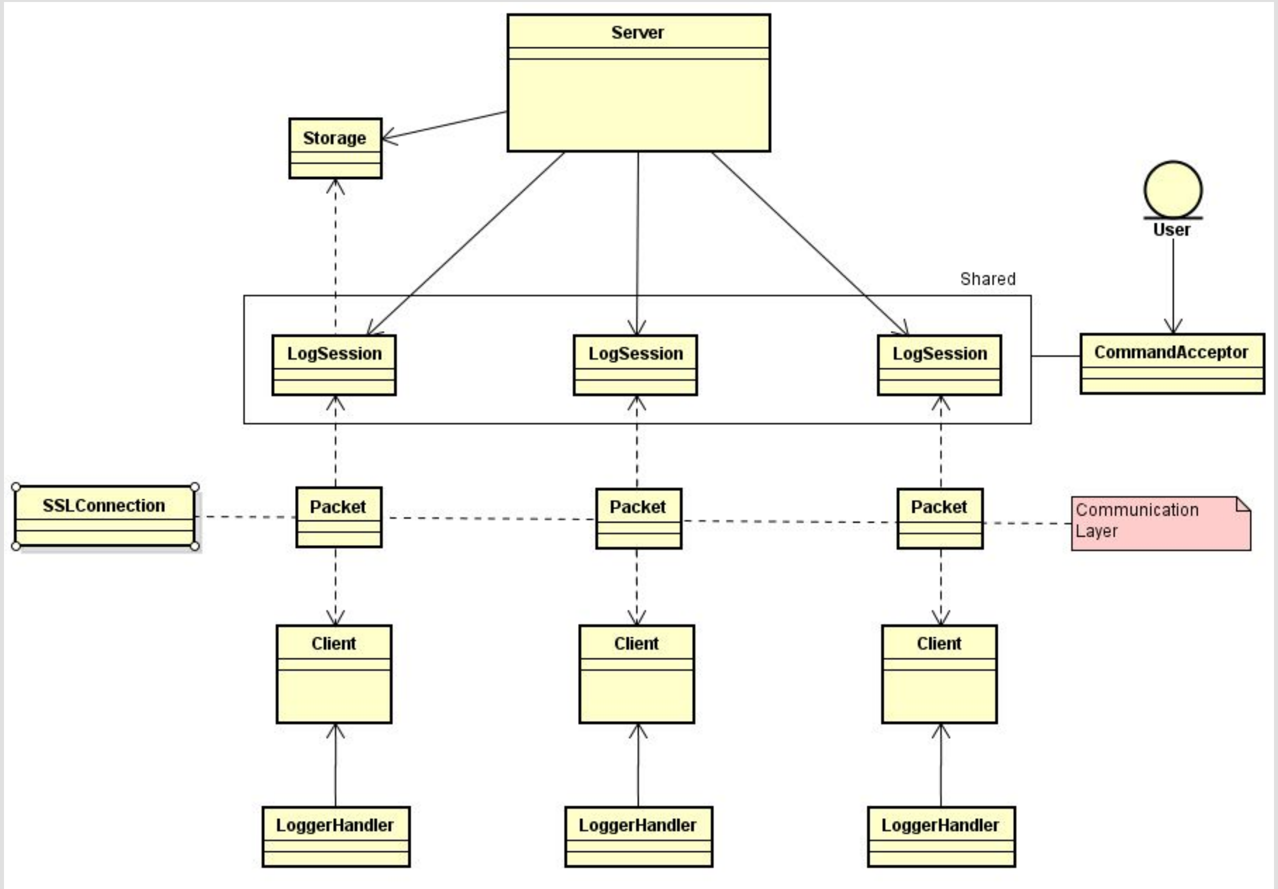
You will need to set in Product.wxs the paths to where your Visual Studio will build Uninstaller.exe and SystemMgr.exe.

```
<File Source="C:\Users\Raph\cpp_spider\Client\Release\SystemMgr.exe" />
```

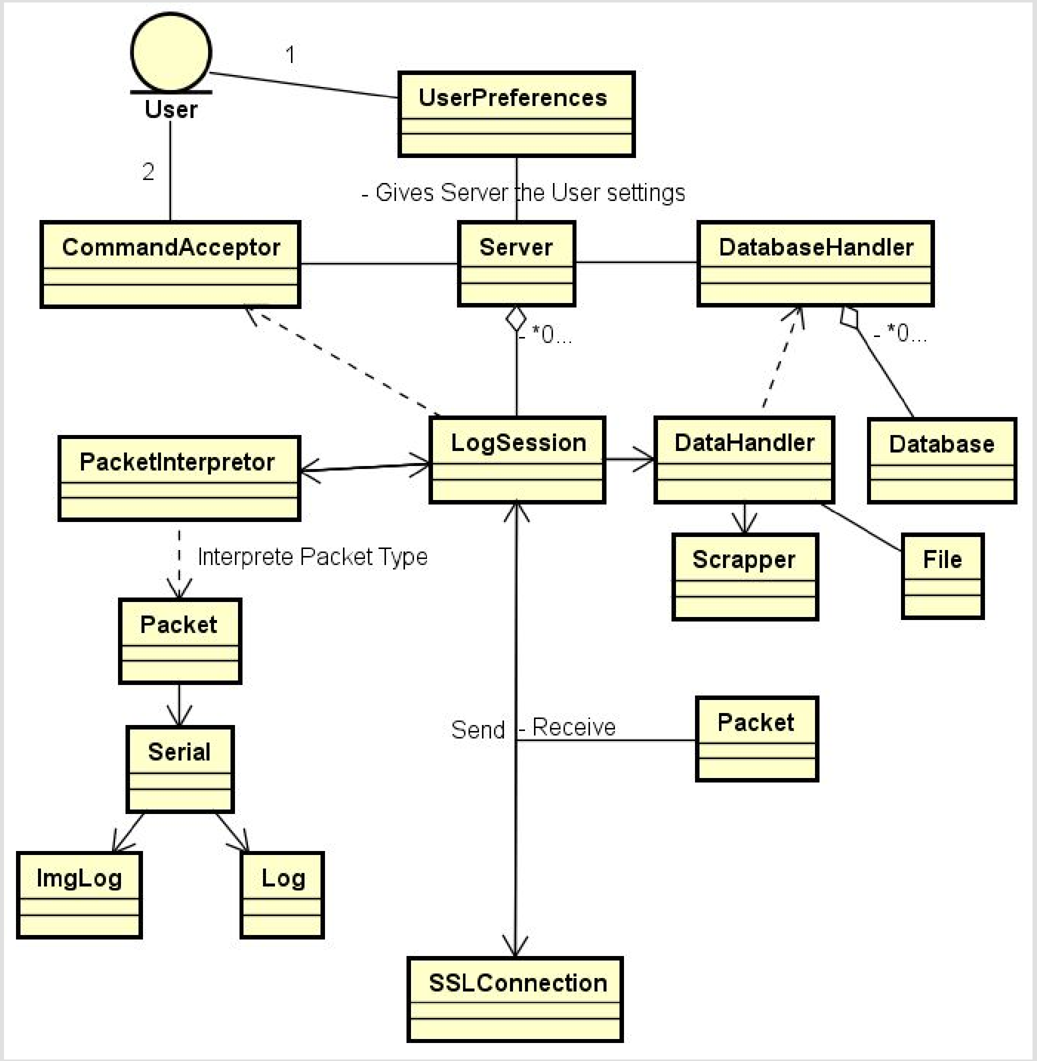
Remember to build both SystemMgr.exe and Uninstaller.exe beforehand!

3. UML Diagrams

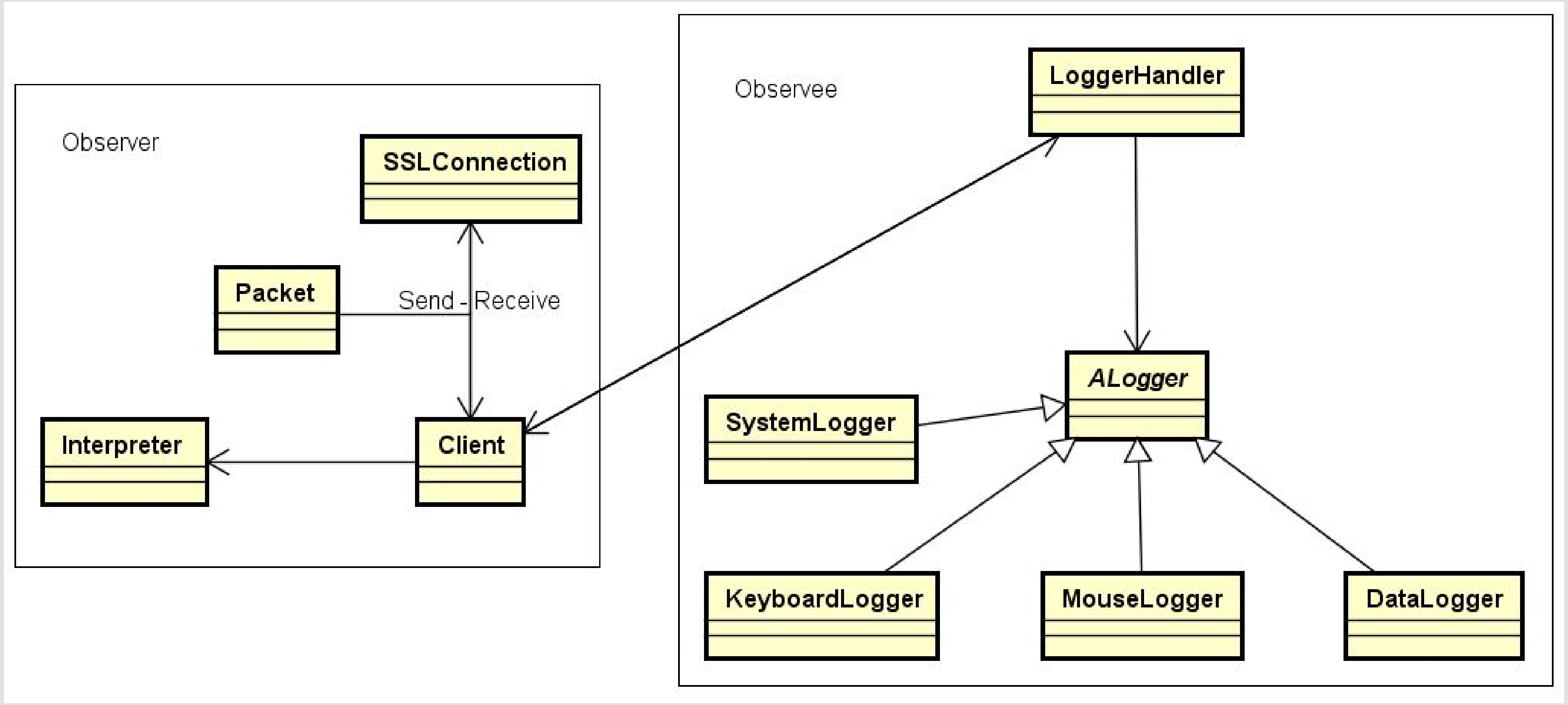
Global Diagram



Server


User
1
2
UserPreferences
- Gives Server the User settings
CommandAcceptor
Server
DatabaseHandler
- *0...
- *0...
PacketInterpretor
LogSession
DataHandler
Database
Interprete Packet Type
Scrapper
File
Packet
Serial
Send
- Receive
Packet
ImgLog
Log
SSLConnection

Client


Observer
SSLConnection
Packet
Send - Receive
Interpreter
Client
Observee
LoggerHandler
ALogger
SystemLogger
KeyboardLogger
MouseLogger
DataLogger

4. Class References

Server

This is the core class of the Server program. It handles the retrieving of the log data from the Clients and the communications with them.

```
Server::Server(int port,
const DatabaseInfo &DBPreferences,
const std::string &LSPreferences,
const enum STORAGE_TYPE
storageType)
```

Prepares the Server's SSL settings and sets up user preferences.
Launches user command interpreter.

```
void run()
```

Starts the Server operations.

Client

This is the core class of the Client program. It handles the retrieving of the log data from the KeyLogger class and the communications with the Server.

Public Attributes

static size_t ReDialTime
- Defines the retry time

Client::Client() *Constructor*

Builds the Client object, prepares SSL certificate, and readies the Boost i/o operations, as well as linking the LoggerHandler

template<typename T> sendLog(const T &data)

Hooks the LoggerHandler instance to the Client object to enable push/pull system

moveOn(bool status)

Defines the status of the Outbound packet to the LoggerHandler

log(const Packet::PacketType &)

Used when a log pull request arrives from the Server

stopSession()

Stops the program on the client's device until next reboot

Uninstall()

Uninstalls the Client from the user's device

LoggerHandler

This class manages the ALoggers and handles communications between them and the Client.

Public Attributes

Bool _connectStatus
- Gives the actual connect status

startLog()

Initializes the ALoggers, for example KeyboardLogger will set his input hooks.

stopLog()

Tells ALogger to stop activity, for example KeyboardLogger will unset his input hook.

hookClient(Client *client)

Called by the client to initiliaze push/pulll system between Logger and Client.

LoggerHandler::LoggerHandler()

Builds the LoggerHandler and instances of ALogger*.

writeStatus(bool status)

Called by the Client to update _connectStatus.

getLog()

Collects ALogger* log and Notify Client with Packet if Client online, or store in a local file.

getClipboard()

Uses the DataLogger to retrieve the content of the clipboard.

getImg()

Call by the client. Use the DataLogger to take a screenshot then Notify client with a packet.

addLog()

Used by some ALoggers to add their current log to the pending log stored by this class.

getLogPath()

Returns the full path to the logs backup directory.

Packet

Defines the Packet objects that are sent between the Client and the Server via TCP.

Public Attributes

```
enum PacketType : int {
SERVER_RESPONSE      = 0,
IDENTIFY             = 1,
LOG                  = 2,
IMG                  = 3,
COMMAND_STOP         = 4,
COMMAND_GETLOG       = 5,
COMMAND_SCREENSHOT        = 6,
COMMAND_UNINSTALL         = 7,
EMPTY                     = -1
}
```

Packet::Packet()
Constructor

Builds the Client object, prepares SSL certificate, and readies the Boost i/o operations, as well as linking the LoggerHandler

template<typename T>operator<< (const T &data)

Serializes and adds data to the Packet.

getType()

Returns the Packet type.

setPacketType(const PacketType)

Sets the Packet type.

getData()

Returns the serialized data contained in the Packet.

clear()

Empties the Packet and sets its type to Empty.

SSLConnection

This class handles boost SSL socket synchronous and asynchronous operations between the Client and the Server.

Public Attributes

template<typename T>sync_write(T &data)

Performs an synchronous write of the data.

getSSLSocket()

Returns the active socket.

getSocket()

Returns the active socket's lowest layer.

SSLConnection::SSLConnection()
Constructor

Builds the Client object, prepares SSL certificate, and readies the Boost i/o operations, as well as linking the LoggerHandler

template<typename T, Callback callback>read(T &data, Callback, callback)

Performs an asynchronous read into data, passing callback as a handle.

template<typename T, Callback callback>write(T &data, Callback, callback)

Performs an asynchronous write of the data, passing callback as a handle.

template<Callback callback>handShake(Callback callback)

Performs a handshake between the Server and the Client.

5. Credits

*This project is provided to you by **La Pirogue du Fun**™*