B5 - Advanced C++ Programming

B-PAV-531

Spider

I spy with my little eye...

42.0

EPITECH.
L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

Spider

binary name:	spider
group size:	4-6
repository name:	cpp_spider
repository rights:	ramassage-tek
language:	C++

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.

This project consists in a keylogger software, using a client/server architecture.

```
A eunuch has no honor, and a spider does not enjoy the luxury of scrupules, my
lord.
    Varys, the Spider.
```

GENERALITIES

+ PLATFORMS

This project is divided in two parts:

- The client MUST run on **Windows**.
- The server MUST run on **Unix**.

On Windows, you MUST use **Microsoft Visual Compiler (MSVC)** to build your project. **Using MinGW is NOT allowed.**

+ PROTOCOL

The **Spider** project aims to create a keylogger "network" with many clients and one or more servers.

As such, you have to come up with a protocol that will allow both transmission of data from the clients to the server, but also the possibility of instructions sent by the server. It MUST be usable over the Internet (you cannot use anything LAN specific).

+ LIBRARIES

Any non-explicitly authorized library is explicitly forbidden. However:

- All client-to-server cocmmunication MUST make use of the **OpenSSL** library.
- You must be able to justify the use of each and every concept you decide to use in your project. Uses that are deemed inapproriate will be punished.
- You are allowed and encouraged to use Boost, both client and server side.
- You must provide an abstraction of every **C** library you decide to use.

Have a look at Boost ASIO for networking.

+ VERSIONING

You SHOULD use code versioning during the entire project.

A clever use of your versioning system WILL be checked during the defense! Use accurate commit logs! Don't mess up your repositories! Unlike during your second year projects, we will take your usage of the versioning system into careful consideration.

MANDATORY PART

+ CONTENTS

The mandatory part of this project consists in all the basics given in the **Basics** sub-section, as well as at least 4 elements from the **Advanced** sub-section.

Any additional item from the **Advanced** sub-section will be considered a bonus.

Any group that implements all the elements from the **Advanced** sub-section will be awarded a full-completion bonus.

Each element from the **Advanced** sub-section can be implemented any way you please

2.1.1 Basics

- A **printed** version of your protocol for your communications (mandatory for **BOTH** follow-ups!)
- A **printed**, fully UML compliant class diagram for **BOTH** client and server (mandatory for **BOTH** follow-ups!)
- A network abstraction implemented in terms of **Boost** or custom-made.
- An encryption abstraction implemented in terms of **OpenSSL**.
- The client MUST be able to log keystrokes locally if there is no network connection or no server available.
- The client MUST routinely check server availability until it connects.
- The client SHOULD record mouse movements, as well as click activity and coordinates.
- The server MUST be able to record the logs received over the network and differentiate between clients.
- The log SHOULD be usable, in the sense that it should be both human-readable and SHOULD reflect what the user typed.
- The client MUST be able to receive commands from the server, allowing it to either change its behavior or report its status.
- It is then logical to assume that the server MUST be able to send commands to clients.

For more information and/or precisions on the way to log keystrokes on Windows

2.1.2 Advanced

- Accurate recording of keystrokes, including (but not limited to):
  - The name of the active process at the time
  - Timestamp of the log
  - Smart logging (ie: using the time between keystrokes to estimate character grouping)
- Advanced encryption through the use of public and private keys.
- Basic viral behavior, including (but not limited to):
  - Starting with Windows
  - Making it impossible for debugging programs to trace program behavior
- Advanced viral behavior, including (but not limited to):
  - Hiding the process from the task manager
  - Hindering anti-virus processes, or at least not being detected
  - Replication
- Pattern detection (server-side) allowing for better logging.
- Use of a real database on the server, to allow for better data exploitation. Your database system/library MUST be brought to your local teacher and approved before the implementation follow-up.
- Make the server a fully autonomous control center, checking on the status of all clients, as well as modifying the behavior of clients on the fly, depending on the client status and/or the last logs they sent.

If you implement a plug-in system and add the bonuses as plug-ins you will get up to 3 extra points

+ Documentation

In addition to all your code and resources, your repository must contain a *"doc"* directory, containing AT LEAST:

- A UML class-diagram in PDF format
- A text file describing your project's Object-Oriented Design. This file should contain any information needed to implement the project according to your design.

To illustrate the quality of your design, we recommend using examples: why is your design good? Does it allow simple modifications? What would I have to do to incorporate a new viral behavior, such as a Wallpaper modifier, using your design? What would I have to do to add a video game to your Spider in which enemies appear every time a keylog is received?

+ STEPS

It is important that **YOU** defend **YOUR** product. The evaluators will only rate what you show them! Organize your follow-ups and defense, share speaking time... Long story short; be **PREPARED**! Don't come as a tourist. It will be the first time for you passing such serious follow-ups and defenses!

Remember that you are rated individually! You've been warned.

Organize your team and affect roles to everyone. We **WILL** rate every aspect of your team, so don't go at it "tech1 style"

- Design follow-up: you must show and defend your design during this follow-up. The two main points that will be heavily discussed are your binary protocol and the design of your client and server. Focus on designing clever abstractions for network, logging... Be smart and creative! The class diagram and protocol description are **MANDATORY** and **MUST** be printed. Other documents such as use cases, sequence diagrams and anything you consider relevant will be very appreciated and will improve our consideration of your seriousness.

We are not talking of your program's graphical design, but rather of your **Object-Oriented Design**, meaning the relationship between your objects, and any **design patterns** you may be inclined to use

- Implementation follow-up: during this follow-up, we will check the quality of your abstractions' implementations and their basic functionalities. A working prototype of your Spider is expected, including (at least) the ability to record keystrokes in an intelligent manner, as well as basic network connectivity. Focus on matching your design and your implementation!
- Final defense: last step of the project, during which we will check the completion of your project and its functionalities. We prefer small and working Spiders to huge but half-finished ones.

Although they are mandatory, the grades for the two follow-ups won't count in your final Spider grade. However, they are a great indicator for you of what we think of your work. Consider these follow-ups a privilege, not a punishment. Be serious and work hard, it'll be worth it.

+ TEAMS

You **MUST** work in team with one other group. As a consequence, you **MUST** team-up with one other **Spider** group and design a binary protocol together.

Most of the points available during the final defense will involve the ability to use your server with the other group's client, and your client with the other group's server.

It is VERY important that you register on the same time slot for the final defense, or we WON'T be able to test your work. Don't mess up. This does not apply to the follow-ups, although it is preferable for both groups to be present to discuss the common design of the binary protocol.

As you may have already understood:

- If one group fails, both fail.
- If one group is late or missing, both fail.
- If one group doesn't pass a test.
- You are responsible of the other group.
- Your final grade is not shared, but it will be heavily impacted by the behavior of the other team.

General setpoints

You are (more or less) free to implement the client and server any way you please. However, here are a few restrictions:

- The only authorized functions from the **libc** are the ones that wrap system calls (and don't have C++ equivalents!)
- Any solution to a problem **MUST** be object-oriented.
- Any not explicitly authorized library is explicitly forbidden.
- Any value passed by copy instead of reference or pointer MUST be justified, or you'll lose points.
- Any member function or method that does not modify the current instance not **const** MUST be justified, or you'll lose points.
- Koalas don't use any **C++** norm. However, any code that is deemed unreadable, unmaintainable or with unnecessary performance costs **WILL** be arbitrarily sanctioned. Be rigorous! Write code you'll be proud of!
- Any conditional branching longer than **if ... else if ... else ...** is FORBIDDEN. Factorize! Use the STL's **associative containers**.
- Keep an eye on this subject regularly, it could be modified.
- We pay great attention to our subjects. If you run into typos, spelling mistakes or inconsistencies, please contact us so we can correct it.
- You can contact the authors by mail. Their adresses can be found on the module's page, under "Module Designers"
- The **C++** Yammer group will contain information and answers to your questions. Please make sure the answer to your question can't be found there before contacting the authors.