Advanced C++ Project

Spider

Koalab koala@epitech.eu
Data Security Hub security@hub.epitech.eu

Abstract: 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 scruples, my lord. Varys, the Spider.

Contents

Chapter I

Generalities

I.1 Platforms

This project is divided in two parts:

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

You are free to use the Windows 7 and OpenSuse systems available to you on your laptops, or choose another platform for your development/defense.

I.2 Protocol

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

As such, you have to write a protocol, allowing both the transmission of data from the client to the server, but also the possibility of server- sent instructions. It MUST be usable over the internet (you cannot use anything LAN specific).

I.3 Libraries

Any non-explicitely allowed library is explicitly forbidden. However :

- All client-server communication MUST make use of the `OpenSSL` library.
- You're completely free to use every bit of C++11 you know of.
- Be careful however, that you are able to justify the uses of each and every concept you decide to use in your project. Uses that are deemed inapropriate will be punished.
- You are allowed and encouraged to use `boost` on the server side and on the client side. (`http://www.boost.org`)
- Every `C` library you decide to use MUST be abstracted.

`Have a look at BOOST Asio at http://www.boost.org/doc/libs/1_40_0/doc/html/boost_asio.html for networking.`

I.4 versioning

You MUST use code versioning during the whole project.

⚠ A clever use of your versioning system WILL be checked during defense! Use precise commit logs! Don't mess up with your repositories! In contrast with your second year, we take your usage of the versioning system into careful consideration.

Chapter II

Mandatory part

II.1 Contents

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

Every supplementary item from the `Advanced` sub-section will be considered a bonus.

Every 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 implemtended any way you choose, following your own interpretation. However you MUST be able to justify your choices.`

II.1.1 Basics

- A **printed** version of your protocol for your communicatons (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 hand made.
- An abstraction hiding your use of the `OpenSSL` library MUST be implemented and present in your conception.
- The client MUST be able to log keystrokes locally, in case there is no network, or no server available.
- However 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 from the network and differentiate between clients.
- The log SHOULD be usable, in the sense that it should be both readable and SHOULD reflect what the user typed.
- The client MUST be able to receive commands from the server allowing it to either change it's behavior or report status of the logging.
- It is then logical to assume that the server MUST be able to send commands to the client, and as such you have to make sure your server will be built for it.

`For more information and/or precisions on the way to log keystrokes on Windows, you can use the following documentation: http://msdn.microsoft.com/en-us/library/windows/desktop/ms644984(v=vs.85).aspx`

II.1.2 Advanced

- Precise record of keystrokes, including (but not limited to):
  - Active processus at the time (where did the user type).
  - Time of the log
  - Smart logging (ie: use time between keystrokes to estimate characters grouping).
- Advanced encryption of the communication through the use of public/private keys.
- Basic viral behavior, including (but not limited to):
  - Start with Windows.
  - Stop debugging programs from tracing the program behavior.
- Advanced viral behavior, including (but not limited to):
  - Hide process from task manager.
  - Hinder anti-virus processes, or at least don't be detected.
  - Replicate.
- Pattern detection (server-side) allowing for better logging.
- Use of a real database on the server, to allow for better exploitation of data. For this your database system/library MUST be brought to us and validated before the implementation follow-up.
- Make the server a fully autonomous control center, checking on the status of all clients, as well as modifying behaviors of clients on the fly depending on the client status/last logs sent.

`If you implement a plug-in system and add the bonuses as plug-ins you will get up to 3 extra points, and a free coffee from your local person in charge.`

II.2 Steps

⚠ It is important that **YOU** defend **YOUR** product. Koalas will rate only what you show them! Organize your follow-ups and defense, share speaking time,...Long story short; be PREPARED! Don't come as a tourist.

⚠ Remember that you are rated individually! This is not bullshit, you've been warned.

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

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

- Implementation follow-up: We will check during this follow up the quality of your abstractions' implementations and their basic functionnalities. A functionnal prototype of Spider is expected, including (at least) recording keystrokes in an intelligent manner, as well as basic network connectivity. Focus on the accuracy between your conception and your implementation!

- Final defense: Last step of the project, we will check the completion of your project and its functionnalities. We prefer small and fully functionnal Spiders to huge but half-finished ones.

II.3 Teams

For the first time you MUST work in team with another group. As a consequence, you MUST team-up with one other `Spider` group and share a commonly designed protocol for your communications.

Most of the points available during the final defense will concern your ability to use your server with your team group client and your client with your team group 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 is not relevant for the follow-ups though.

As you may have already understood:

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

Chapter III

General setpoints

You are (more or less) free to do the implementation you want. However, here are a few restrictions.

- The only allowed 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 allowed library is explicitly forbidden.
- Any value passed by copy instead of reference or pointer MUST be justified, otherwise you'll lose points.
- Any non-`const` value passed as parameter MUSt be justified, otherwise you'll lose points.
- Any member function or method that does not modify the current instance not `const` MUST be justified, otherwise you'll lose points.
- Koalas don't use any `C++` norm. However, any code that we deem unreadable or ugly can be arbitrarily sanctioned. Be rigorous!
- Any conditional branching longer than `(if ... else if ... else ...)` is FORBIDDEN! Factorize!
- Keep an eye on this subject regularly, it could be modified.
- We pay a great attention to our subjects, if you run into typos, spelling mistakes or inconsistencies, please contact us at `koala@epitech.eu` so we can correct it.
- You can contact the authors by mail, see the header.
- The intranet's forum will contain informations and answers to your questions. Be sure the answer to you question is not already there before contacting the authors.

Chapter IV

Delivery setpoints

You MUST deliver your project on the BLIH repository made available for you by the Bocal. Your repository's name MUST be cpp_spider.

We will ignore any commit which happened after the project's end. Complaining that "it wasn't 11:42 pm on my watch" is useless. The only time that counts for us is Epitech's time.

Murphy's law corollary : "If you deliver your work during the last hour, something's gonna go wrong.".

Only the code on your repository will be evaluated during the defense.

Good luck !