Project #1-A

Computer and Network Security — Released: 17Aug2021
COMP-5370/-6370 — Due: 25Aug2021 at 6pm CT

Part A of this project is due **25Aug2021 at 1800 CT** and you may choose to use your "late days" as discussed in the syllabus. The final deadline is 27Aug2021 at 1800CT (using both) and any submission after that time will be a zero (0). Submissions will be accepted through Canvas prior to the initial deadline or by emailing to the TA between the initial and final deadline.

This portion of the project must be completed individually

Overview

In order to effectively evaluate security and privacy properties of technology, it is important to not only analyze what *should* happen but also what *could* happen. An excellent microcosm of this scenario is code which serializes and deserializes (sometimes called marshalling/unmarshalling) which has long been a source of bugs and exploitable vulnerabilities (Python, PHP, Android, etc.). This is further complicated by overly complex file formats which may provide arbitrary functionality through such mechanisms as JavaScript or macros.

For this project, you will develop software which requires you to think both as the *builder* (defensively) and the *breaker* (offensively). The specification for your software is an entirely made-up data format named `nosj` and is available along with this assignment. You are also being provided with a framed-out Python3 module and a set of initial test-cases. Based only on the specification and these test-cases, you should be able to complete this portion of the project in its entirety. You are only required to use the module containing the Python3 exceptions (`exceptions.py`) but it is **highly recommended** that you build off of the pre-existing code in order to avoid "re-implementing the wheel":

serialization.py + deserialization.py These are frame-out modules for Part 1 and Part 2 respectively. The auto-grader will use the module names and function templates to test your submission.

exceptions.py This module contains the two exceptions which you **MUST** use in your implementation (see Restrictions below).

test_implementation.py A ready-built test-harness for both Part 1 and Part 2. This harness uses the pytest framework and all tests can be executed by running the command `pytest`.

Setup

For this project, you will use Python3's built-in `venv` package to isolate your environment and avoid interfering with any other Python dependencies you may have. Follow the below steps for macOS/Linux. Windows is analogous and described in the module documentation.

- Create and move to the directory where you are going to work
  - `mkdir ./project-1a`
  - `cd project-1a`
- Create your virtual environment
  - `python3 -m venv py-env`

- Load your virtual environment
  - `source py-env/bin/activate`
- Install the dependencies
  - `pip3 install -U pytest`
- Copy the framed-out files above into your current directory.
  - `wget https://comp5370.org/proj1a-framing.tar.gz`
  - `tar -xf proj1a-framing.tar.gz`
- Run tests
  - `pytest`
  - Tests will fail due to not being implemented but will confirm that your environment is setup correctly.

⋆⋆⋆ DON'T FORGET TO ACTIVATE ENVIRONMENT BEFORE RUNNING THE TESTS ⋆⋆⋆

Part 1: Build-It - Serialization

You will implement a Python3 module capable of marshalling a Python3 dictionary into the nosj format. The `serialization.py` file already contains some framing that may be useful (though you are *not* required to use it).

Part 2: Build-It - Deserialization

You will implement a Python3 module capable of unmarshalling a nosj string into a Python3 dictionary. The `deserialization.py` file already contains some framing that may be useful (though you are *not* required to use it).

Restrictions

1. Your implementation **may not** rely on the Python2 interpreter

   Python2 is dangerous and wholly unacceptable.

2. Your implementation **may not** raise standard Python3 exceptions.

   You **are** allowed to raise the `SerializationError` and `DeserializationError` exceptions contained in the `exceptions.py` module to indicate that a problem is unrecoverable. These exceptions are expected to have a descriptive and useful message attached to them.

3. Your implementation **may not** "catch" any exception thrown (commonly implemented via built-in exception handling using the `try-except-finally` pattern. This includes your own exceptions via the `exceptions.py` module (i.e. you can't raise-catch your own exceptions).

   You should *validate* the input rather error-and-recover.

4. Your implementation **may not** make use of resources not self-contained in the Python3 interpreter with the exception of the `pytest` module as used above.

   If your approach requires external logic, you are almost certainly violating the KISS principle of engineering.

Submission Details

Various expectations of your submission are listed below and **they are non-negotiable**. If you submission fails to meet these expectations, you may receive a zero (0) for the entire project. If you have any questions, about submission format, details, contents, or anything discussed below, seek clarification rather than making assumptions.

Expectations:

- It must contain two and **only** two files (`serialization.py` and `deserialization.py`).
- Each file must contain the same externally-available functions as the framing in order to be graded
- Your implementation **must not** do anything other than marshal/unmarshal to the specification provided. The explicitly dis-allowed functionality includes, but not limited to:
  - Writing directly to a file
  - Reading anything including files, variables, configurations, etc
  - Making network requests
  - Attempting to install packages
- No information should be printed to `stdout` or `stderr`
- Your submission must be named "$< last - name >$`-project-1a.tar.gz`" and be a gzip'd tarball
  - `tar -czf name-project-1a.tar.gz serialization.py deserialization.py`
  - Zipfiles **are not** acceptable
- The files must be in the root of the tarball and not nested in a directory

Grading

Submissions will be graded based on:

- Test-cases provided with the framed-out code
- Private test-cases used only for grading
- Maintainability of the code to other developers
  - This is largely that it is readable, understandable, and modifiable by a person other than the one who wrote it
- Meeting the expectations listed above