Computer and Network Security

COMP-5370/-6370/-6376

Homework #4

This homework is due **24Nov2020 at 1800 CT** and you may choose to use your "late days" as discussed in the syllabus. The final deadline is 27Nov2020 at 1800CT (using all three) and any submission afterwards will be a zero (0). If you choose to use a late day, you must alert the TA of that fact *before* the deadline.

You may work on this homework solo or in groups of no more than two. It is entirely your choice but the responsibility of forming, communicating, coordinating, and collaborating is yours as well. The groups may be across any sections of the course and are not restricted by in-person or distance-learning aspects. Piazza is an excellent way to find other people who wish to work in a group. In your submission to Canvas, you must include a `team-members.txt` file which lists them group members' names and their email addresses **even if there is only one team member**.

Overview

Serialization and deserialization has long been a source of bugs which can be turned into exploitable vulnerabilities (Python, PHP, Android, etc.). This is further complicated by overly complex file formats which may running JavaScript internally (link) or trying to allow arbitrarily complex macros (link). Although there have been substantial efforts to improve the current state, the amount of legacy code and the widespread use of certain less-than-ideal "features" has caused many to re-evaluate whether a clean-slate approach is needed.

In this homework, you will explore a real-world scenario in which you must think offensively and defensively in order succeed. The specification for the completely made-up data format named nosj is available in Canvas and you are being provided with a set of framed 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 project in its entirety but additional test-cases may be provided to clarify common questions or highlight specific aspects.

Existing Codebase

A number of files are posted to Canvas which will be useful in getting you started on this homework. You are only required to use one of them (`exceptions.py`) but it is **highly recommended** that you at least explore them to see if you can avoid re-implementing the wheel.

serialization.py and deserialization.py — These are frame-out modules for Part 1 and Part 2 respectively. You are not required to use them in any way whatsoever.

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, we will be using Python3's built-in venv package to isolate. To setup your environment follow the below steps for macOS/Linux. Windows is analogous and included with the rest of the venv module documentation.

- Create and move to the directory where you are going to work.
- `python3 -m venv py-env`
- `source py-env/bin/activate`
- `pip3 install -U pytest`
- Copy the framed-out files above into your current directory.
- `pytest`
  - Tests will fail but ensures that you have installed everything correct.

If you've never worked with Python3 virtual-environments before, I suggest you skim over the documentation but it boils down to a couple of straight-forward things:

- Any modules installed via pip/pip3 in that venv only are accessible while in that venv.
- You have to explicitly enter a venv to run code inside of it via the `source` command above.
- You have to explicitly leave a venv to return to your normal terminal environment via the `deactivate` command.

Restrictions

This project was initially designed for a type-unsafe language such as C or C++ but has been converted to use Python3 to make it less troublesome given the short amount of time. That being said, there are certain restrictions placed on it to mimic some canonical limitations of those languages:

1. Your implementation may **not** raise standard Python3 exceptions and any where are raised will be treated as a hard-crash of the internal implementation. 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.
2. You are **not** allowed to use Python3's try-except-finally construction. This includes standard Python3 exceptions as well as those in the `exceptions.py` module (i.e. you can't raise-catch your own exceptions).
3. You are **not** allowed to use any module that is not built-in to Python3. The sole exception is the `pytest` module if you choose to write your own test-cases.

Part 1: Build-It - Serialization

For this portion of the homework, 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). If you choose not to, make sure and double-check the "What to Submit" section to ensure that your solution is properly structured.

Part 2: Build-It - Deserialization

For this portion of the homework, 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). If you choose not to, make sure and double-check the "What to Submit" section to ensure that your solution is properly structured.

Bonus: Break-It (up to 20 points)

For this portion of the homework, you will create two test-cases for each of the above parts (four total) which are specifically constructed to trick other students' implementations. It is *highly recommended* that you leverage your time, energy, and efforts on Parts 1/2 of this homework to make this portion less time-intensive. While implementing your (de)serialization modules, keep an eye-out for unexpected corner-cases or tricky implementation details that you encounter. Entire classes of bugs exist because independent developers make identical mistakes due to a variety of reasons.

Submission Details

Various expectations of your Canvas submission are listed below. **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, ask on Piazza. If you believe there is an error in the below requirements, contact Dr. Springall as soon as possible.

File uploaded to Canvas:

- It must be named `homework-4.tar.gz`.
  - Canvas will auto-rename it but this should be the name of the file you upload to Canvas.
- It may contain `breaker.py` if you are trying for the bonus points.
- It must contain:
  - `serialization.py`
  - `deserialization.py`
  - The `team-members.txt` file containing the names and emails of the team members you worked in a group with. **This file must be included even if you worked solo.**
- The above files must be in the root of the tarball and not nested in a directory.
- It must not contain any file other than those listed above including, but not limited to `test_implementation.py`, `exceptions.py`, or any other source file.
  - You can list the structure and contents of a tarball with the Bash command:
    `tar -tf homework-4.tar.gz`
- It must be a gzip'd tarball
  - Zipfiles are **not** acceptable.

The easiest way to create your submission file is with the Bash command:

```
tar -czf homework-4.tar.gz serialization.py deserialization.py breaker.py team-members.py
```

Specific to serialization.py Your module **MUST** contain an exposed `marshal()` function which takes-in a Python3 dictionary and one of:

- Returns a Python3 string object of the nosj serialization
- Throws a `SerializationError` with a descriptive error

Specific to deserialization.py Your module **MUST** contain an exposed `unmarshal()` function which takes-in a Python3 string representing the nosj marshalled-state and either:

- Returns a Python3 dictionary of the same data structure
  - **This dictionary must have correct Python3-typed elements.**
- Throws a `DeserializationError` with a descriptive error

Specific to breaker.py Your module **MUST** contain the following exposed variables:

- **marshal_ok_input** — A Python3 dictionary that will be sent to the `marshal()` function of other modules.
- **marshal_ok_expect** — A Python3 string of the correct response your `marshal_ok_input` object.
- **marshal_bad_input** — A Python3 dictionary that will be sent to the `marshal()` function of other modules. It is expected that a correctly implemented module will raise an `SerializationError` exception.
- **unmarshal_ok_input** — A Python3 string that will be sent to the `unmarshal()` function of other modules.
- **unmarshal_ok_expect** — A Python3 dictionary of the correct response your `unmarshal_ok_input` object.
- **unmarshal_bad_input** — A Python3 string that will be sent to the `unmarshal()` function of other modules. It is expected that a correctly implemented module will raise an `DeserializationError` exception.

In addition to the above variables, your module should include documentation that answers the following questions in 1-3 sentences each:

- What specific corner-case is the input attempting to trigger?
- What is the correct way to handle that corner-case?
- Why do you expect other developers to incorrectly handle that corner-case?

For all submitted .py files:

- They must be Python3 scripts.
  - Python2 is dangerous and wholly unacceptable.
- They must be organized, cleanly written, and the logic must be explicitly clear from the source.
  - Remember that descriptive function names and variables go a long way towards making code readable.

  - You are encouraged to document any portions of your source code that may not be clear to the average developer.

- With the exception of importing the `exceptions.py` module and potentially the `pytest` module, they must not use any Python package that is not in the Python3 standard library.

- They must not do anything other than marshal/unmarshal to the specification. 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

- It must not output any debugging information or other information.