Final Report: Stability and Correctness of Python's `pickle` Module

---

Team Members and Contributions

- **Xiuyu Dong**: Lead Researcher – Designed test suite, coordinating black-box testing.
- **Tianzhuo Wu**: Develeoper - White-box testing code implementation and documentation.
- **Chengyang Luo**:Documentation Assistant – Helped organize test results, formatted the final report, and contributed to writing the findings and conclusion sections.
- **Renhao Qian**: Quality Assurance & Security Analyst – Conducted Bandit security analysis on the pickle module, interpreted the vulnerability reports, and contributed to the security discussion and risk assessment section.

1. Introduction

The `pickle` module in Python is widely used for serializing and deserializing Python objects. However, its determinism—whether identical inputs always produce identical serialized outputs—has been a topic of discussion. This report investigates the stability and correctness of `pickle`, focusing on whether identical inputs consistently yield identical serialized outputs across different environments and conditions.

According to it's own document, `pickle` module is not secure as it will execute arbitrary code during unpickling.

During realworld conditions, the `pickle` module would be used (by its own words) pickling and unpickling. Which is a serializing and deserializing process which is similar to what would be known as `json`, what is the difference except for the safety concerns is that after serialization, `pickle` would generate the content that is not human-readable,

while the `json` would generate what human can read.

Using bandit do detect the vulnerability of the `pickle` , stats as follows.

```
Test results:
>> Issue: [B101:assert_used] Use of assert detected. The enclosed code will be removed when compiling to optimised byte code.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b101_assert_used.html
   Location: ./pickle_main_file.py:507:8
506                 return
507             assert id(obj) not in self.memo
508             idx = len(self.memo)
```

```
>> Issue: [B101:assert_used] Use of assert detected. The enclosed code will be removed when compiling to optimised byte code.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b101_assert_used.html
   Location: ./pickle_main_file.py:791:8
790             # without memoizing them
791             assert self.proto >= 3
792             n = len(obj)
```

```
>> Issue: [B101:assert_used] Use of assert detected. The enclosed code will be removed when compiling to optimised byte code.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b101_assert_used.html
   Location: ./pickle_main_file.py:817:8
816             # without memoizing them
817             assert self.proto >= 5
818             n = len(obj)
```

```
>> Issue: [B101:assert_used] Use of assert detected. The enclosed code will be removed when compiling to optimised byte code.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b101_assert_used.html
   Location: ./pickle_main_file.py:1256:16
1255                    raise EOFError
1256                assert isinstance(key, bytes_types)
1257                dispatch[key[0]](self)
```

```
>> Issue: [B101:assert_used] Use of assert detected. The enclosed code will be removed when compiling to optimised byte code.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b101_assert_used.html
   Location: ./pickle_main_file.py:1802:4
1801        res = f.getvalue()
1802        assert isinstance(res, bytes_types)
1803        return res
```

2. Test Suite Design

2.1. Testing Techniques Applied

- **Equivalence Partitioning**: Categorized inputs into distinct classes to ensure comprehensive coverage.
- **Boundary Value Analysis**: Tested edge cases such as empty objects and deeply nested structures.

- **Fuzz Testing**: Introduced random variations to inputs to identify potential inconsistencies.

2.2. Test Cases

The test suite included the following categories:

- **Primitive Data Types**: Integers, floats, strings, booleans.
- **Collections**: Lists, tuples, sets, dictionaries.
- **Custom Classes**: User-defined classes with various attributes.
- **Recursive Structures**: Objects containing references to themselves.
- **Cross-Version Serialization**: Objects serialized in one Python version and deserialized in another.
- **Cross-Platform Serialization**: Objects serialized on one operating system and deserialized on another.
- Big Ints and long floats: Integars which bigger than the int class, float which need more precision than the float class.

3. Traceability Matrix

REQUIREMENT ID	TEST CASE ID	DESCRIPTION	STATUS
R1	TC1	Serialization of integers	Passed
R2	TC2	Serialization of strings	Passed
R3	TC3	Serialization of user-defined classes	Passed(?)
R4	TC4	Serialization of recursive structures	Failed
R5	TC5	Cross-version serialization	Passed
R6	TC6	Cross-platform serialization	Passed
R7	TC7	Same content with different types	Failed
R8	TC8	Empty Object	Passed(?)

4. White-box Testing Module

To gain a deeper understanding of the internal mechanisms of the `pickle` module, we designed and executed white-box tests focusing on behaviors related to object referencing (memoization), recursive structures, nested data structures, and serialization support for functions and lambdas.

TEST ASPECT	CODE FUNCTION
Memoization Behavior	`test_memo_behavior()`
Recursive Structures	`test_recursive_structure()`
Large Number of Unique Objects	`test_large_unique_objects()`
Nested Structures	`test_nested_structure()`
Function and Lambda Serialization	`test_pickle_function_lambda()`

*__Memoization Behavior Test__: Verified that when the same object is referenced multiple times, `pickle` uses memoization to reuse object references, reducing the size of serialized data. Results showed significantly smaller data size for repeated references of the same instance, confirming the effectiveness of memoization.

*__Recursive Structure Test__: Constructed self-referential lists to test if `pickle` can correctly serialize and deserialize recursive object structures. The recursive structures were restored properly without stack overflow or exceptions.

*__Large Number of Unique Objects Test__: Serialized a list containing over 10,000 unique custom objects to test `pickle`'s stability when handling large volumes of objects. Serialization completed successfully.
*__Nested Structure Test__: Tested serialization and deserialization of deeply nested data structures containing custom objects, confirming data consistency and correctness.

*__Function and Lambda Serialization Test__: Confirmed that `pickle` has limited support for serializing functions and lambda expressions, with serialization attempts failing as expected.

This white-box testing module enhances our understanding of `pickle`'s internal behavior.

5. Findings

- **Primitive Data Types**: Serialization of basic data types (integers, strings, etc.) was deterministic and consistent across environments.
- **Recursive Structures**: Serialization of recursive structures was consistent, but issues arose when combined with custom classes.
- **Cross-Version Serialization**: Objects serialized in one Python version and deserialized in another often resulted in errors or inconsistencies due to changes in the `pickle` protocol.
- **Cross-Platform Serialization**: Serialization across different operating systems led to discrepancies, likely due to differences in object memory layouts and internal representations.
- When it comes to the Empty Object and Custom classes, it is interesting that the python output false while pickle output true.

6. Discussion

6.1. Reasons for Non-Determinism

Factors might contribute to the non-deterministic behavior of `pickle`:

- **Object Memory Addresses**: The memory address of an object can vary between sessions, affecting its serialized representation.
- **Internal State Variations**: Differences in internal states, such as reference counts or object IDs, can lead to different serialized outputs.

6.2. Limitations of the Test Suite

- **Scope**: The test suite focused on a subset of Python's data types and structures; other types may exhibit different behaviors.
- **Environment Variability**: Differences in hardware and Python configurations were not exhaustively tested.
- **Complexity of Objects**: Highly complex or non-standard objects were not included in the test suite.

7. Recommendations

- **Use Alternative Serialization Formats**: For applications requiring deterministic serialization, consider using formats like JSON or Protocol Buffers, which are designed for portability and consistency.
- **Avoid Pickling Functions**: Functions and lambdas are not reliably serializable with `pickle` and may lead to inconsistencies.

8. Conclusion

The `pickle` module does not guarantee deterministic serialization across different environments and conditions. While it is suitable for short-term storage and inter-process communication within controlled environments.

9. References

- Python 3.12.7 Documentation: pickle — Python object serialization
- Stack Overflow Discussion on Pickle Determinism

10. Repository

The source code for the test suite and additional documentation can be found in the following repository:

GitHub Repository Link