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

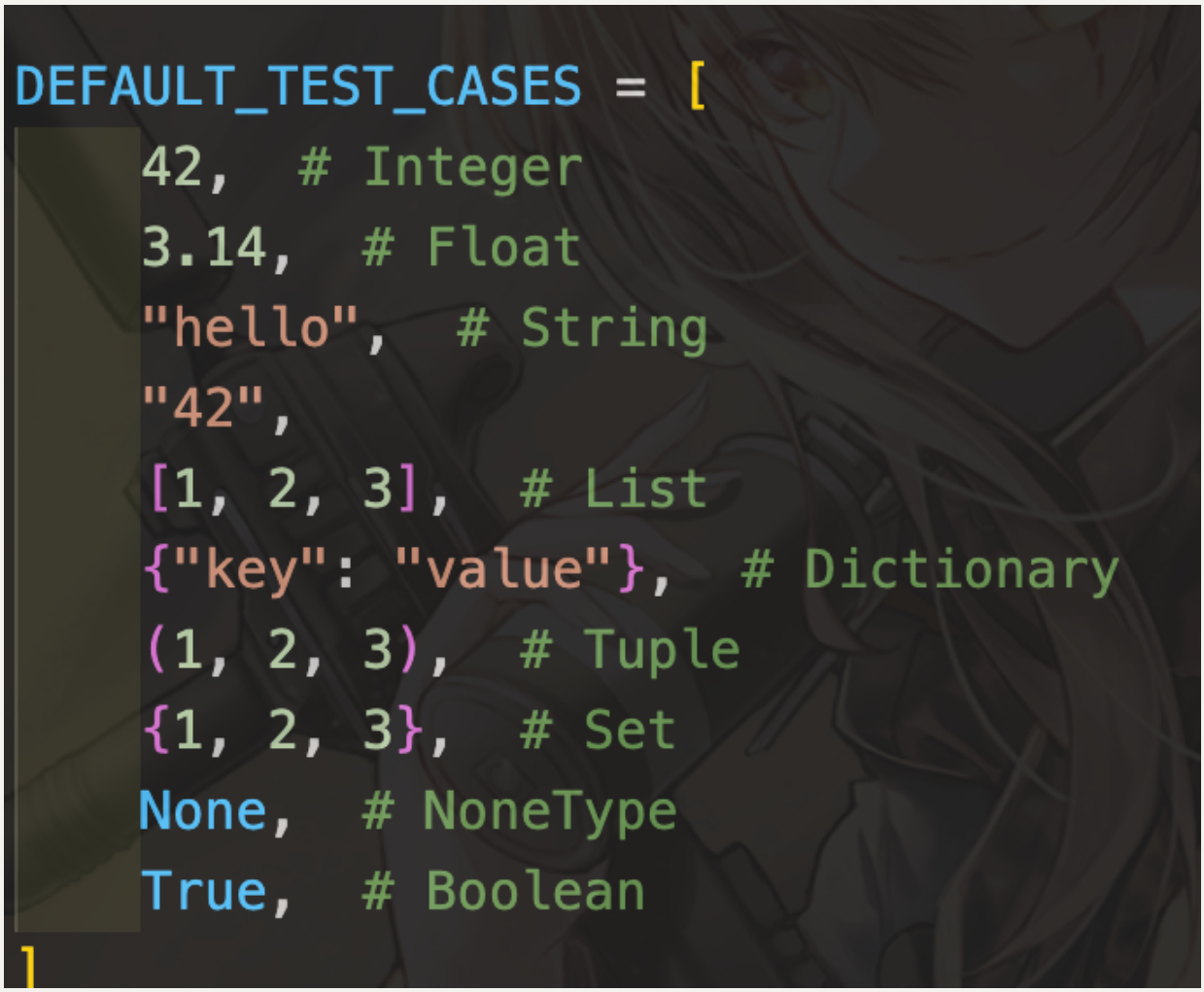
- **Use Alternative Serialization Formats**: For applications requiring deterministic serialization, consider using formats like JSON or Protocol Buffers, which are designed for portability and consistency.
- **Avoid Pickling Functions**: Functions and lambdas are not reliably serializable with `pickle`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

8. Conclusion

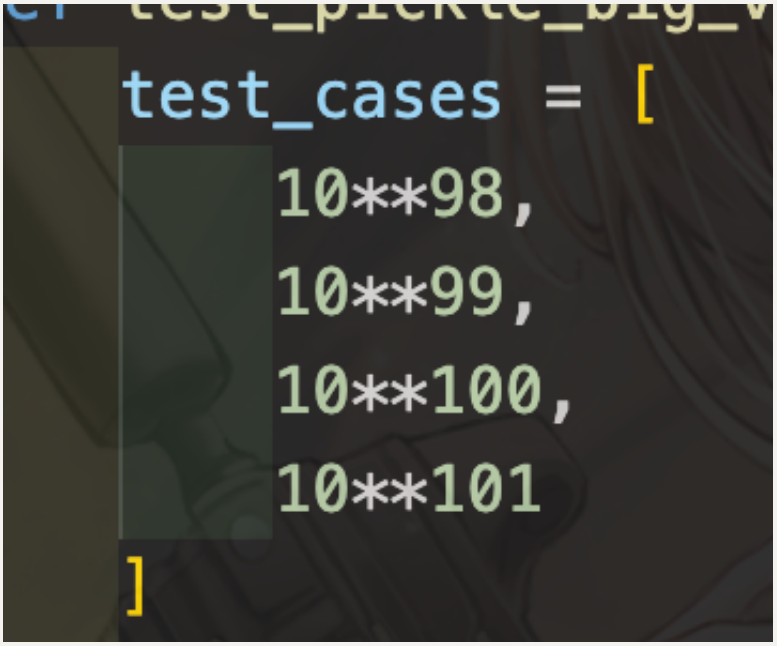
The `pickle` module does not guarantee deterministic serialization across different environments and conditions. While it is suitable for short-term storage and inter-process communication within controlled environments.

9. References

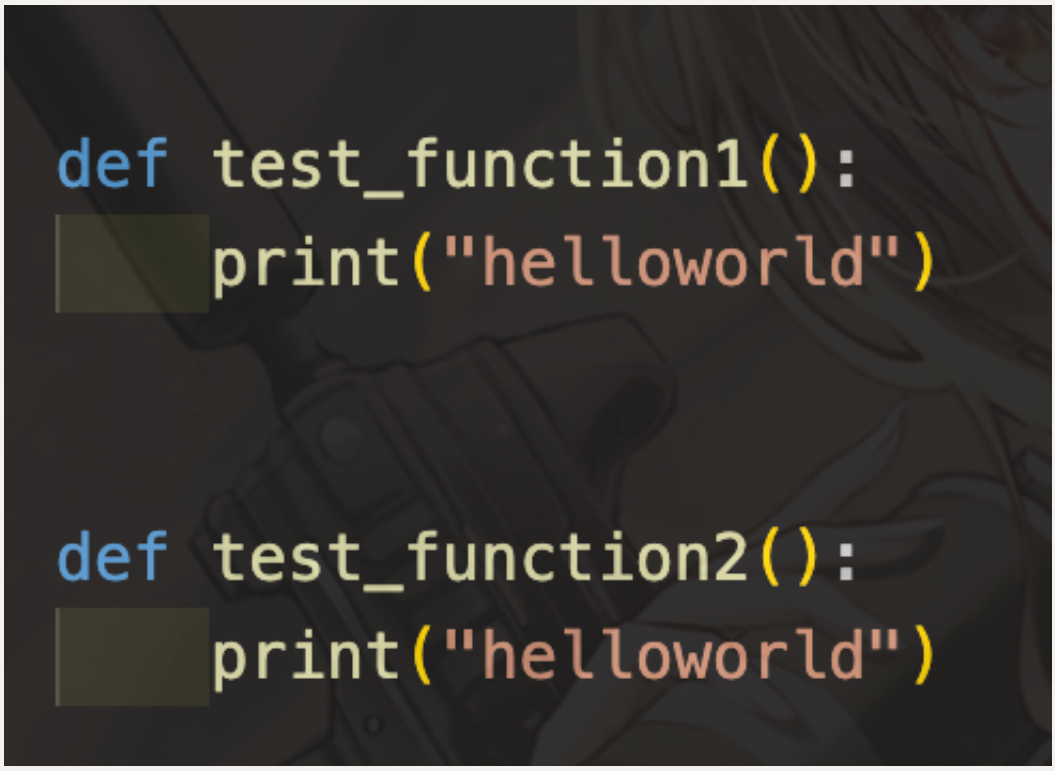
- Python 3.12.7 Documentation: pickle — Python object serialization
- Stack Overflow Discussion on Pickle Determinism
- Used datasets:



```
DEFAULT_TEST_CASES = [
    42,  # Integer
    3.14,  # Float
    "hello",  # String
    "42",
    [1, 2, 3],  # List
    {"key": "value"},  # Dictionary
    (1, 2, 3),  # Tuple
    {1, 2, 3},  # Set
    None,  # NoneType
    True,  # Boolean
]
```



```
    test_cases = [
        10**98,
        10**99,
        10**100,
        10**101
    ]
```




```python
# 测试大量不同对象（触发 memo 扩容）
# Test serialization with many unique objects to stress memo table
# 对应白盒测试方法：数据流测试、循环覆盖
# White-box method: Data Flow Testing, Loop Coverage
def test_large_unique_objects():
    print("\nTest: large number of unique objects")
    try:
        data = pickle.dumps([TestObject(i) for i in range(10000)])  # 大量不同对象
        print("Large object list serialized successfully")
    except Exception as e:
        print("Serialization failed:", e)
```

```python
# 测试 Pickler 的 memo 行为：同一对象两次序列化应复用 memo 引用
# Test Pickler's memo behavior: same instance should be referenced via memo
# 对应白盒测试方法：条件覆盖、路径覆盖
# White-box method: Condition Coverage, Path Coverage
def test_memo_behavior():
    print("Test: memo behavior")
    obj = TestObject(123)
    data1 = pickle.dumps([obj, obj])  # 同一对象引用两次，应使用 memo 优化
    data2 = pickle.dumps([TestObject(123), TestObject(123)])  # 值相等但为两个对象，memo 不复用

    print("Same instance size:", len(data1))
    print("Different instance size:", len(data2))
    assert len(data1) < len(data2), "Memoization should reduce size"
```

```python
# 嵌套结构测试 Unpickler 栈的使用是否正确
# Test nested structure serialization/deserialization
# 对应白盒测试方法：条件覆盖、语句覆盖
# White-box method: Condition Coverage, Statement Coverage
def test_nested_structure():
    print("\nTest: nested structure")
    obj = {'a': [1, 2, {'b': TestObject(99)}]}  # 多层嵌套结构
    data = pickle.dumps(obj)
    restored = pickle.loads(data)
    assert restored['a'][2]['b'].value == 99
    print("Nested structure restored successfully")
```

```python
# 测试函数和 lambda 是否报错（pickle 不支持）
# Test that pickling unsupported objects (functions/lambdas) raises an error
# 对应白盒测试方法：异常路径覆盖、判定覆盖
# White-box method: Exception Path Coverage, Decision Coverage
def test_pickle_function_lambda():
    print("\nTest: pickling function or lambda")

    def f():
        return 1

    try:
        pickle.dumps(f)
        print("Function pickled (unexpected)")
    except Exception as e:
        print("Function pickling failed as expected")

    try:
        pickle.dumps(lambda x: x + 1)
        print("Lambda pickled (unexpected)")
    except Exception as e:
        print("Lambda pickling failed as expected")
```

```python
# 测试递归结构处理（Unpickler 栈是否正常清空）
# Test recursive structures: checks for stack handling during (de)serialization
# 对应白盒测试方法：路径覆盖、循环覆盖
# White-box method: Path Coverage, Loop Coverage
def test_recursive_structure():
    print("\nTest: recursive structure")
    x = []
    x.append(x)  # 递归引用自身
    try:
        data = pickle.dumps(x)
        result = pickle.loads(data)
        assert result[0] is result, "Recursive structure restored incorrectly"
        print("Recursive test passed")
    except Exception as e:
        print("Recursive test failed:", e)
```

etc.

with The test results screenshots below.

```
All original tests passed, Storing All Hash256 in File
['81976fef9fe34f8f64469792f27360d11781b927280419c5f5514c0da0235b54', '7a8161062076a8001170b2d5be1fcd933570f86733e7bd69fcaceeb46e495ea5', '4
25c7a325bad320fe5ab4ba6cfb411d9b79168bc7a3e2f9b4428b88f876ba7f1', 'f4ff434b109d0f34a33ebfd9330bdf5e007b5cb971dc7a6285397bc3f6308585', 'f934
3d7d7ec5c3d8bcced056c438fc9f1d3819e9ca3d42418a40857050e10e20', '06abbdd67a6a827cdf824746600e46776fe4231d33c4da6f963756e44eec7719', '8fb4a92
569cb17a1480e9415543642bd724725e2a6e3ecf85a8e2b8c7729d30f', '844e80e1f1ffaba7019345c229a5b8599bdb348864e993f1cbdc2962ce8e23ce', '9c298d589a
2158eb513cb52191144518a2acab2cb0c04f1df14fca0f712fa4a1', '112bda3b495d867b6a98c899fac7c25eb60ca4b6e6fe5ec7ab9299f93e8274bc']
All float tests passed
test passed with 15 zeros, it can't handle more due to the precision problem of float class.
Failed for object: 100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
Failed for object: 1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
Failed for object: 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
Failed for object: 100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
Big int test failed, it can handle above 128bits numbers.
Different types value different
8a349cb1febd1789f57975b1d2ba936f8e9a13eb3480d38fce6beeb03fced711
c012125d2616ea6958f83b2a9685b1dae711224e226bd65a325e7450bed698dd
Time based tests passed.
Test failed with functions with different names and the same content
Test failed with functions with same names and the same content but different packages
False
test passed
False
test passed
```

```
All original tests passed, Storing All Hash256 in File
['81976fef9fe34f8f64469792f27360d11781b927280419c5f5514c0da0235b54', '7a8161062076a8001170b2d5be1fcd933570f86733e7bd69fcaceeb46e495ea5', '4
25c7a325bad320fe5ab4ba6cfb411d9b79168bc7a3e2f9b4428b88f876ba7f1', 'f4ff434b109d0f34a33ebfd9330bdf5e007b5cb971dc7a6285397bc3f6308585', 'f934
3d7d7ec5c3d8bcced056c438fc9f1d3819e9ca3d42418a40857050e10e20', '06abbdd67a6a827cdf824746600e46776fe4231d33c4da6f963756e44eec7719', '8fb4a92
569cb17a1480e9415543642bd724725e2a6e3ecf85a8e2b8c7729d30f', '844e80e1f1ffaba7019345c229a5b8599bdb348864e993f1cbdc2962ce8e23ce', '9c298d589a
2158eb513cb52191144518a2acab2cb0c04f1df14fca0f712fa4a1', '112bda3b495d867b6a98c899fac7c25eb60ca4b6e6fe5ec7ab9299f93e8274bc']
Compare the file if you want, system version and python version.

For instance, input windows first, press enter, then input 3.12.3 .
If you don't want to do that, input QUIT. All uppercase.
Linux
Linux
3.9.21
3.9.21
Linux3921.csv
Test passed for object on different systems and different python versions.
```

```
Test: memo behavior
Same instance size: 61
Different instance size: 72

Test: recursive structure
Recursive test passed

Test: large number of unique objects
Large object list serialized successfully

Test: nested structure
Nested structure restored successfully

Test: pickling function or lambda
Function pickling failed as expected
Lambda pickling failed as expected
```

10. Repository

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

GitHub Repository Link