TRAIL OF BITS

The Dos and Don’ts of testing

Test Your Tests

phaze
@lovethewired

@trailofbits

Blockchain Team

Overview

- **Motivation for topic**
- **Examples: Testing shortcomings**
- **Exploring various testing strategies**
- **Takeaways**

The Role of Testing

- **Fault identification**
- **Invariant validation**
- **Spec adherence**
- **Build up confidence and trust in performance**
- **Guarding code functionality: Regression tests**

But: No testing method is foolproof.

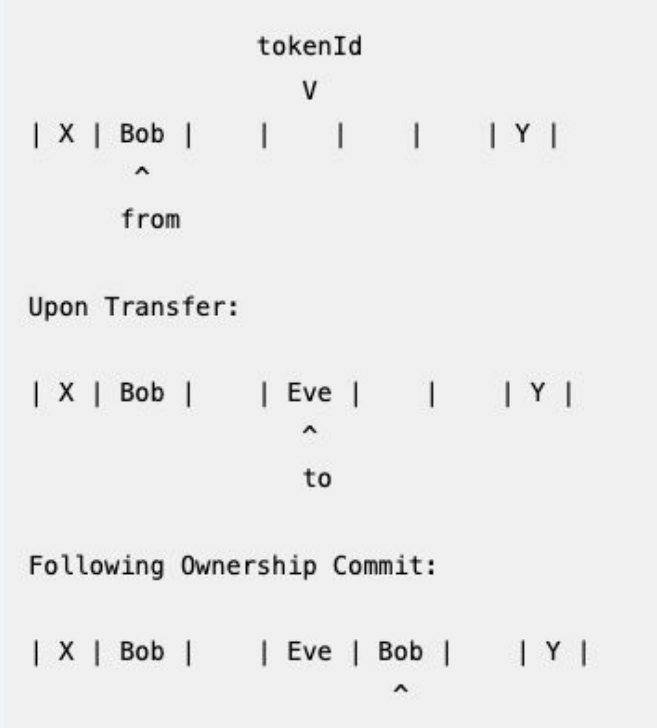
Testing is an ongoing process of refinement, not a final endpoint!

Motivation for Topic

- **Reflections on past shortcomings**
- **Improving development and testing process**
- **Driver for becoming security-oriented**

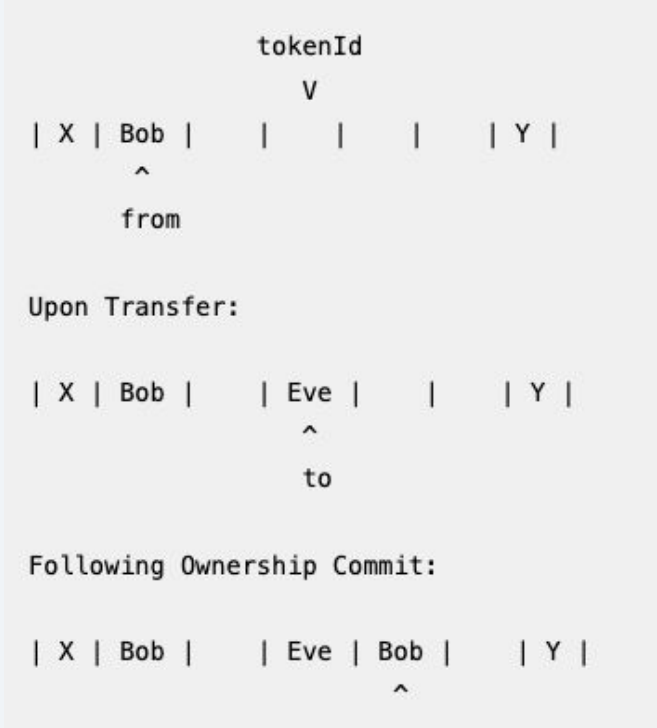
Improving ERC721A

ERC721A Optimization



ERC721A Optimization

- **Save gas on all future transfers**
- **Store boolean `nextTokenDataSet` in ownership slot**
- **Only touch subsequent token data if `!nextTokenDataSet`**



Development & Testing Approach

- **Test-driven**
- **“Sufficient testing will uncover flaws”**
- **Quality through quantity**

But: Lost sight of the bigger picture

Unit-Testing Functionality

```
/// Transfer last minted id.
/// Ensure id beyond last is not written to.
/// Ensure correct ownership.
function test_transferFrom1() public {
    // ...
}

/// Transfer last two minted ids in batch.
/// Ensure correct ownership.
function test_transferFrom2() public {
    // ...
}

/// Mint new tokens after previous transfers.
/// Ensure correct ownership.
function test_transferFrom3() public {
    // ...
}

/// Transfer tokens in middle of batch.
/// Ensure correct ownership.
function test_transferFrom4() public {
    // ...
}
```

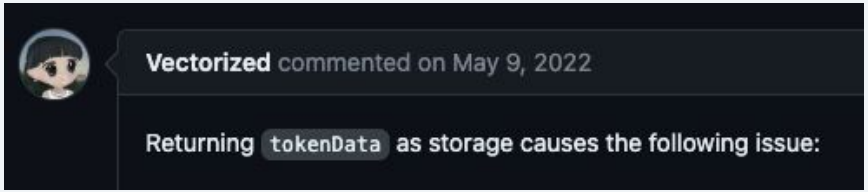

Unit-Testing Functionality

```
/// Transfer last minted id.
/// Ensure id beyond last is not written to.
/// Ensure correct ownership.
function test_transferFrom1() public {
    // ...
}

/// Transfer last two minted ids in batch.
/// Ensure correct ownership.
function test_transferFrom2() public {
    // ...
}

/// Mint new tokens after previous transfers.
/// Ensure correct ownership.
function test_transferFrom3() public {
    // ...
}

/// Transfer tokens in middle of batch.
/// Ensure correct ownership.
function test_transferFrom4() public {
    // ...
}
```



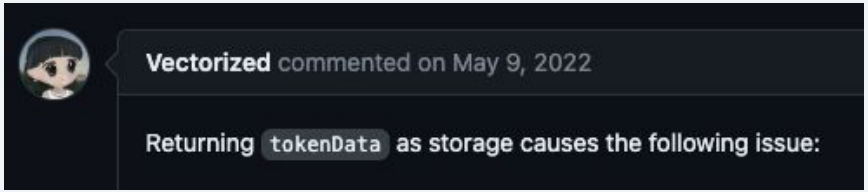
Unit-Testing Functionality

```
/// Transfer last minted id.
/// Ensure id beyond last is not written to.
/// Ensure correct ownership.
function test_transferFrom1() public {
    // ...
}

/// Transfer last two minted ids in batch.
/// Ensure correct ownership.
function test_transferFrom2() public {
    // ...
}

/// Mint new tokens after previous transfers.
/// Ensure correct ownership.
function test_transferFrom3() public {
    // ...
}

/// Transfer tokens in middle of batch.
/// Ensure correct ownership.
function test_transferFrom4() public {
    // ...
}
```



How??!

Apply Patch and Test

```
    function transferFrom(
        address from,
        address to,
        uint256 tokenId
    ) public {
        // Get token data. This might be implicit and
        // thus stored at an id other than `tokenId`.
        TokenData storage tokenData = _tokenDataOf(tokenId);

        // ...

-       TokenData storage actualTokenData = _tokenData[tokenId];
-       actualTokenData.owner = to;
-       actualTokenData.nextTokenDataSet = true;
+       tokenData.owner = to;

        if (!tokenData.nextTokenDataSet) {
+           tokenData.nextTokenDataSet = true;
            // If the ownership slot of `tokenId + 1` is not explicitly set, then `from` owns it.
            // Set the slot of `tokenId + 1` explicitly in storage to maintain correctness.
            TokenData storage nextTokenData = _tokenData[tokenId + 1];
            if (nextTokenData.owner == address(0) && tokenId + 1 < _currentIndex) {
                nextTokenData.owner = from;
            }
        }
    }
```

```
/// Transfer first token in batch.
/// Ensure correct ownership.
function test_transferFrom5() public {
    token.mint(bob, 10);

    vm.prank(bob);
    token.transferFrom(bob, alice, 1);

    assertEq(token.ownerOf(1), alice);
    assertEq(token.ownerOf(2), bob);
}
```

Fixed!!
All tests pass…

Apply Patch and Test

```
    function transferFrom(
        address from,
        address to,
        uint256 tokenId
    ) public {
        // Get token data. This might be implicit and
        // thus stored at an id other than `tokenId`.
        TokenData storage tokenData = _tokenDataOf(tokenId);

        // ...

-       TokenData storage actualTokenData = _tokenData[tokenId];
-       actualTokenData.owner = to;
-       actualTokenData.nextTokenDataSet = true;
+       tokenData.owner = to;

        if (!tokenData.nextTokenDataSet) {
+           tokenData.nextTokenDataSet = true;
            // If the ownership slot of `tokenId + 1` is not explicitly set, then `from` owns it.
            // Set the slot of `tokenId + 1` explicitly in storage to maintain correctness.
            TokenData storage nextTokenData = _tokenData[tokenId + 1];
            if (nextTokenData.owner == address(0) && tokenId + 1 < _currentIndex) {
                nextTokenData.owner = from;
            }
        }
    }
```

```
/// Transfer first token in batch.
/// Ensure correct ownership.
function test_transferFrom5() public {
    token.mint(bob, 10);

    vm.prank(bob);
    token.transferFrom(bob, alice, 1);

    assertEq(token.ownerOf(1), alice);
    assertEq(token.ownerOf(2), bob);
}
```

Fixed!! (Not quite)
All tests pass…

What Went Wrong?

- **Lacking systematic testing approach and structure**

What Went Wrong?

- **Lacking systematic testing approach and structure**
- **Missing important edge-cases**

What Went Wrong?

- **Lacking systematic testing approach and structure**
- **Missing important edge-cases**
- **Testing multiple things at once**

What Went Wrong?

- **Lacking systematic testing approach and structure**
- **Missing important edge-cases**
- **Testing multiple things at once**
- **Lacking expressive and meaningful fuzz tests**
  - Multiple transfers
  - Random ids
  - Arbitrary actors

What Went Wrong?

- **Lacking systematic testing approach and structure**
- **Missing important edge-cases**
- **Testing multiple things at once**
- **Lacking expressive and meaningful fuzz tests**
  - Multiple transfers
  - Random ids
  - Arbitrary actors

100% code line & branch coverage != 100% state coverage

Good Testing is Hard...

Shortcomings Exemplified: Testing WAD Conversions

Testing WAD Conversions

```
uint256 constant WAD = 1e18;

function fromWAD(uint256 a) pure returns (uint256 c) {
    c = a / WAD;
}

function toWAD(uint256 a) pure returns (uint256 c) {
    c = a * WAD;
}

contract TestFromWAD_DONT is Test {
    function test_toWAD_fromWAD(uint256 a) external {
        if (a >= type(uint256).max / WAD) vm.expectRevert();

        assertEq(fromWAD(toWAD(a)), a);
    }
}
```

Testing WAD Conversions

```
uint256 constant WAD = 1e18;

function fromWAD(uint256 a) pure returns (uint256 c) {
    c = a / WAD;
}

function toWAD(uint256 a) pure returns (uint256 c) {
    c = a * WAD;
}

contract TestFromWAD_DONT is Test {
    function test_toWAD_fromWAD(uint256 a) external {
        if (a >= type(uint256).max / WAD) vm.expectRevert();

        assertEq(fromWAD(toWAD(a)), a);
    }
}
```

```
Running 1 test for test/TestFromWAD.sol:TestFromWAD_DONT
[PASS] test_toWAD_fromWAD(uint256) (runs: 10000, μ: 1236, ~: 589)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 439.95ms
```


Testing WAD Conversions (DON'T)

```
uint256 constant WAD = 1e18;

function fromWAD(uint256 a) pure returns (uint256 c) {
    c = a / WAD;
}

function toWAD(uint256 a) pure returns (uint256 c) {
    c = a * WAD;
}

contract TestFromWAD_DONT is Test {
    function test_toWAD_fromWAD(uint256 a) external {
        if (a >= type(uint256).max / WAD) vm.expectRevert();

        assertEq(fromWAD(toWAD(a)), a);
    }
}
```

```
Running 1 test for test/TestFromWAD.sol:TestFromWAD_DONT
[PASS] test_toWAD_fromWAD(uint256) (runs: 10000, μ: 1236, ~: 589)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 439.95ms
```

```
Running 2 tests for test/TestFromWAD.sol:TestFromWAD_DONT
[PASS] test_toWAD_fromWAD(uint256) (runs: 10000, μ: 1225, ~: 589)
[FAIL. Reason: Call did not revert as expected] test_toWAD_fromWAD_boundary() (gas: 4093)
Traces:
  [4093] TestFromWAD_DONT::test_toWAD_fromWAD_boundary()
    ├─ [3446] TestFromWAD_DONT::test_toWAD_fromWAD(115792089237316195423570985008687907853269984
665640564039457 [1.157e59])
    │   ├─ [0] VM::expectRevert()
    │   │   └─ ← ()
    │   └─ ← ()
    └─ ← "Call did not revert as expected"

Test result: FAILED. 1 passed; 1 failed; 0 skipped; finished in 217.74ms

Ran 2 test suites: 2 tests passed, 1 failed, 0 skipped (3 total tests)

Failing tests:
Encountered 1 failing test in test/TestFromWAD.sol:TestFromWAD_DONT
[FAIL. Reason: Call did not revert as expected] test_toWAD_fromWAD_boundary() (gas: 4093)
```


Testing WAD Conversions (DON'T)

```
uint256 constant WAD = 1e18;

function fromWAD(uint256 a) pure returns (uint256 c) {
    c = a / WAD;
}

function toWAD(uint256 a) pure returns (uint256 c) {
    c = a * WAD;
}

contract TestFromWAD_DONT is Test {
    function test_toWAD_fromWAD(uint256 a) external {
        if (a >= type(uint256).max / WAD) vm.expectRevert();

        assertEq(fromWAD(toWAD(a)), a);
    }
}
```

```
Running 1 test for test/TestFromWAD.sol:TestFromWAD_DONT
[PASS] test_toWAD_fromWAD(uint256) (runs: 10000, μ: 1236, ~: 589)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 439.95ms
```

```
Running 2 tests for test/TestFromWAD.sol:TestFromWAD_DONT
[PASS] test_toWAD_fromWAD(uint256) (runs: 10000, μ: 1225, ~: 589)
[FAIL. Reason: Call did not revert as expected] test_toWAD_fromWAD_boundary() (gas: 4093)
Traces:
  [4093] TestFromWAD_DONT::test_toWAD_fromWAD_boundary()
    ├─ [3446] TestFromWAD_DONT::test_toWAD_fromWAD(115792089237316195423570985008687907853269984
665640564039457 [1.157e59])
    │   ├─ [0] VM::expectRevert()
    │   │   └─ ← ()
    │   └─ ← ()
    └─ ← "Call did not revert as expected"

Test result: FAILED. 1 passed; 1 failed; 0 skipped; finished in 217.74ms

Ran 2 test suites: 2 tests passed, 1 failed, 0 skipped (3 total tests)

Failing tests:
Encountered 1 failing test in test/TestFromWAD.sol:TestFromWAD_DONT
[FAIL. Reason: Call did not revert as expected] test_toWAD_fromWAD_boundary() (gas: 4093)
```


Testing WAD Conversions (DON'T)

```
uint256 constant WAD = 1e18;

function fromWAD(uint256 a) pure returns (uint256 c) {
    c = a / WAD;
}

function toWAD(uint256 a) pure returns (uint256 c) {
    c = a * WAD;
}

contract TestFromWAD_DONT is Test {
    function test_toWAD_fromWAD(uint256 a) external {
        if (a > type(uint256).max / WAD) vm.expectRevert();

        assertEq(fromWAD(toWAD(a)), a);
    }
}
```

```
Running 1 test for test/TestFromWAD.sol:TestFromWAD_DONT
[PASS] test_toWAD_fromWAD(uint256) (runs: 10000, μ: 1236, ~: 589)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 439.95ms
```

```
Running 2 tests for test/TestFromWAD.sol:TestFromWAD_DONT
[PASS] test_toWAD_fromWAD(uint256) (runs: 10000, μ: 1225, ~: 589)
[FAIL. Reason: Call did not revert as expected] test_toWAD_fromWAD_boundary() (gas: 4093)
Traces:
  [4093] TestFromWAD_DONT::test_toWAD_fromWAD_boundary()
    ├─ [3446] TestFromWAD_DONT::test_toWAD_fromWAD(1157920892373161954235709850086879078532699846
656540564039457 [1.157e59])
    │   ├─ [0] VM::expectRevert()
    │   │   └─ ← ()
    │   └─ ← ()
    └─ ← "Call did not revert as expected"

Test result: FAILED. 1 passed; 1 failed; 0 skipped; finished in 217.74ms

Ran 2 test suites: 2 tests passed, 1 failed, 0 skipped (3 total tests)

Failing tests:
Encountered 1 failing test in test/TestFromWAD.sol:TestFromWAD_DONT
[FAIL. Reason: Call did not revert as expected] test_toWAD_fromWAD_boundary() (gas: 4093)
```


Testing WAD Conversions (DON'T)

```
uint256 constant WAD = 1e18;

function fromWAD(uint256 a) pure returns (uint256 c) {
    c = a / WAD;
}

function toWAD(uint256 a) pure returns (uint256 c) {
    c = a * WAD;
}

contract TestFromWAD_DONT is Test {
    function test_toWAD_fromWAD(uint256 a) external {
        if (a > type(uint256).max / WAD) vm.expectRevert();

        assertEq(fromWAD(toWAD(a)), a);
    }
}
```

```
Running 1 test for test/TestFromWAD.sol:TestFromWAD_DONT
[PASS] test_toWAD_fromWAD(uint256) (runs: 10000, μ: 1236, ~: 589)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 439.95ms
```

- **Know your tool!**

Testing WAD Conversions (DON'T)

```
uint256 constant WAD = 1e18;

function fromWAD(uint256 a) pure returns (uint256 c) {
    c = a / WAD;
}

function toWAD(uint256 a) pure returns (uint256 c) {
    c = a * WAD;
}

contract TestFromWAD_DONT is Test {
    function test_toWAD_fromWAD(uint256 a) external {
        if (a > type(uint256).max / WAD) vm.expectRevert();

        assertEq(fromWAD(toWAD(a)), a);
    }
}
```

```
Running 1 test for test/TestFromWAD.sol:TestFromWAD_DONT
[PASS] test_toWAD_fromWAD(uint256) (runs: 10000, μ: 1236, ~: 589)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 439.95ms
```

- **Know your tool!**
- **Don't solely rely on one type of tests**

Test WAD Conversions (DO)

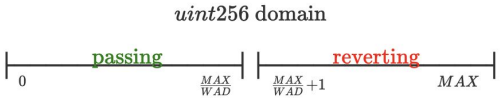
```
contract TestFromWAD_DO is Test {
    /// Testing for values where `toWAD(a)` shouldn't revert.
    function test_toWAD_fromWAD(uint256 a) public {
        a = bound(a, 0, type(uint256).max / WAD);

        assertEq(fromWAD(toWAD(a)), a);
    }

    /// Testing for values where `toWAD(a)` should revert.
    function test_toWAD_fromWAD_revert_Panic(uint256 a) public {
        a = bound(a, type(uint256).max / WAD + 1, type(uint256).max);

        vm.expectRevert(abi.encodeWithSignature("Panic(uint256)", (0x11)));

        assertEq(fromWAD(toWAD(a)), a);
    }
}
```



- **Know your tool!**
- **Don’t solely rely on one type of tests**
- **Restructure tests**
  - **Split tests by outcome/behavior**

Test WAD Conversions (DO)

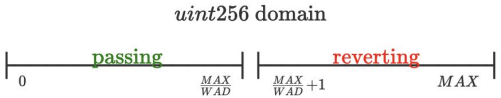
```
contract TestFromWAD_DO is Test {
    /// Testing for values where `toWAD(a)` shouldn't revert.
    function test_toWAD_fromWAD(uint256 a) public {
        a = bound(a, 0, type(uint256).max / WAD);

        assertEq(fromWAD(toWAD(a)), a);
    }

    /// Testing for values where `toWAD(a)` should revert.
    function test_toWAD_fromWAD_revert_Panic(uint256 a) public {
        a = bound(a, type(uint256).max / WAD + 1, type(uint256).max);

        vm.expectRevert(abi.encodeWithSignature("Panic(uint256)", (0x11)));

        assertEq(fromWAD(toWAD(a)), a);
    }
}
```



- **Know your tool!**
- **Don’t solely rely on one type of tests**
- **Restructure tests**
  - **Split tests by outcome/behavior**
  - **Ensure coverage around boundary points**

Test WAD Conversions (DO)

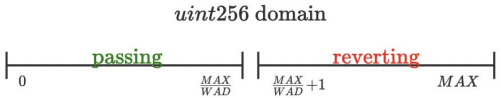
```
contract TestFromWAD_DO is Test {
    /// Testing for values where `toWAD(a)` shouldn't revert.
    function test_toWAD_fromWAD(uint256 a) public {
        a = bound(a, 0, type(uint256).max / WAD);

        assertEq(fromWAD(toWAD(a)), a);
    }

    /// Testing for values where `toWAD(a)` should revert.
    function test_toWAD_fromWAD_revert_Panic(uint256 a) public {
        a = bound(a, type(uint256).max / WAD + 1, type(uint256).max);

        vm.expectRevert(abi.encodeWithSignature("Panic(uint256)", (0x11)));

        assertEq(fromWAD(toWAD(a)), a);
    }
}
```



- **Know your tool!**
- **Don’t solely rely on one type of tests**
- **Restructure tests**
  - **Split tests by outcome/behavior**
  - **Ensure coverage around boundary points**
  - **Reduce complex decision trees**

Test WAD Conversions (DO)

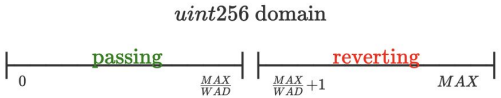
```
contract TestFromWAD_DO is Test {
    /// Testing for values where `toWAD(a)` shouldn't revert.
    function test_toWAD_fromWAD(uint256 a) public {
        a = bound(a, 0, type(uint256).max / WAD);

        assertEq(fromWAD(toWAD(a)), a);
    }

    /// Testing for values where `toWAD(a)` should revert.
    function test_toWAD_fromWAD_revert_Panic(uint256 a) public {
        a = bound(a, type(uint256).max / WAD + 1, type(uint256).max);

        vm.expectRevert(abi.encodeWithSignature("Panic(uint256)", (0x11)));

        assertEq(fromWAD(toWAD(a)), a);
    }
}
```



- **Know your tool!**
- **Don’t solely rely on one type of tests**
- **Restructure tests**
  - **Split tests by outcome/behavior**
  - **Ensure coverage around boundary points**
  - **Reduce complex decision trees**
  - **Expect specific revert**

Shortcomings Exemplified: Testing WAD Multiplication

Testing WAD Multiplication (DON'T)

```
contract TestMulWAD_DONT is Test {
    function mulWAD(uint256 a, uint256 b) public pure returns (uint256 c) {
        unchecked {
            if (b == 0 && a > type(uint256).max / b) revert Overflow();

            c = a * b / WAD;
        }
    }

    function test_mulWAD(uint256 a, uint256 b) public {
        if (b == 0 && a > type(uint256).max / b) vm.expectRevert();

        uint256 c = mulWAD(a, b);

        assertEq(c, a * b / WAD);
    }
}
```

```
Running 1 test for test/TestMulWAD.sol:TestMulWAD_DONT
[PASS] test_mulWAD(uint256,uint256) (runs: 10000, μ: 1730, ~: 755)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 214.91ms
```


Testing WAD Multiplication

```
contract TestMulWAD_DONT is Test {
    function mulWAD(uint256 a, uint256 b) public pure returns (uint256 c) {
        unchecked {
            if (b == 0 || a > type(uint256).max / b) revert Overflow();

            c = a * b / WAD;
        }
    }

    function test_mulWAD(uint256 a, uint256 b) public {
        if (b == 0 || a > type(uint256).max / b) vm.expectRevert();

        uint256 c = mulWAD(a, b);

        assertEq(c, a * b / WAD);
    }
}
```


Testing WAD Multiplication

```
contract TestMulWAD_DONT is Test {
    function mulWAD(uint256 a, uint256 b) public pure returns (uint256 c) {
        unchecked {
            if (b == 0 || a > type(uint256).max / b) revert Overflow();

            c = a * b / WAD;
        }
    }

    function test_mulWAD(uint256 a, uint256 b) public {
        if (b == 0 || a > type(uint256).max / b) vm.expectRevert();

        uint256 c = mulWAD(a, b);

        assertEq(c, a * b / WAD);
    }
}
```

```
Running 1 test for test/TestMulWAD.sol:TestMulWAD_DONT
[PASS] test_mulWAD(uint256,uint256) (runs: 10000, μ: 1730, ~: 755)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 214.91ms
```

Testing WAD Multiplication (DON'T)

```
contract TestMulWAD_DONT is Test {
    function mulWAD(uint256 a, uint256 b) public pure returns (uint256 c) {
        unchecked {
            if (b == 0 || a > type(uint256).max / b) revert Overflow();

            c = a * b / WAD;
        }
    }

    function test_mulWAD(uint256 a, uint256 b) public {
        if (b == 0 || a > type(uint256).max / b) vm.expectRevert();

        uint256 c = mulWAD(a, b);

        assertEq(c, a * b / WAD);
    }
}
```

```
Running 1 test for test/TestMulWAD.sol:TestMulWAD_DONT
[PASS] test_mulWAD(uint256,uint256) (runs: 10000, μ: 1730, ~: 755)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 214.91ms
```

Testing WAD Multiplication (DON'T)

```
contract TestMulWAD_DONT is Test {
    function mulWAD(uint256 a, uint256 b) public pure returns (uint256 c) {
        unchecked {
            if (b == 0 && a > type(uint256).max / b) revert Overflow();

            c = a * b / WAD;
        }
    }

    function test_mulWAD(uint256 a, uint256 b) public {
        if (b == 0 && a > type(uint256).max / b) vm.expectRevert();

        uint256 c = mulWAD(a, b);

        assertEq(c, a * b / WAD);
    }
}
```

```
Running 1 test for test/TestMulWAD.sol:TestMulWAD_DONT
[PASS] test_mulWAD(uint256,uint256) (runs: 10000, μ: 1730, ~: 755)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 214.91ms
```


Exploring Various Testing Strategies (DO)

```
contract TestMulWAD_DO is Test {
    /// Unit test
    function test_mulWAD_unit() public {
        uint256 a = 0;
        uint256 b = 0;
        uint256 c = mulWAD(a, b);

        assertEq(c, 0);
    }
}
```

- **Include unit tests for special cases**

Exploring Various Testing Strategies (DO)

```
contract TestMulWAD_DO is Test {
    // Fuzz test commutative property
    function test_mulWAD_fuzz_commutative(uint256 a, uint256 b) public {
        // If `a * b = c` doesn't overflow...
        try this.mulWAD(a, b) returns (uint256 c) {
            // then `b * a = c`.
            assertEq(c, mulWAD(b, a));
        } catch {}
    }
}
```

- **Include unit tests for special cases**
- **Fuzz test multiple properties**

Exploring Various Testing Strategies (DO)

```
contract TestMulWAD_DO is Test {
    /// Fuzz test
    function test_mulWAD_fuzz(uint256 a, uint256 b) public {
        unchecked {
            uint256 c = a * b;

            if (a != 0 && b != c / a) vm.expectRevert(Overflow.selector);

            assertEq(mulWAD(a, b), c);
        }
    }
}
```

- **Include unit tests for special cases**
- **Fuzz test multiple properties**
- **Re-implement logic from a different angle**

Exploring Various Testing Strategies (DO)

```
contract TestMulWAD_DO is Test {
    function mulWADNative(uint256 a, uint256 b) public pure returns (uint256 c) {
        c = a * b / WAD;
    }

    /// Differential test
    function test_mulWAD_differential(uint256 a, uint256 b) public {
        assertEqCall(
            address(this), //
            abi.encodeCall(this.mulWAD, (a, b)),
            abi.encodeCall(this.mulWADNative, (a, b))
        );
    }
}
```

- **Include unit tests for special cases**
- **Fuzz test multiple properties**
- **Re-implement logic from a different angle**
- **Use differential fuzzing**

Key Takeaways

- **Treat your tests as production code**

Key Takeaways

- **Treat your tests as production code**
- **Understand limitations of testing and tooling**

Key Takeaways

- **Treat your tests as production code**
- **Understand limitations of testing and tooling**
- **Explore different testing strategies and techniques**

Key Takeaways

- **Treat your tests as production code**
- **Understand limitations of testing and tooling**
- **Explore different testing strategies and techniques**
- **Examine assumptions, preconditions, and conclusions of tests**

Key Takeaways

- **Treat your tests as production code**
- **Understand limitations of testing and tooling**
- **Explore different testing strategies and techniques**
- **Examine assumptions, preconditions, and conclusions of tests**
- **Test your tests**

Offensive Testing
Case Study:
Primitive Finance - Hyper

Primitive Finance - Hyper

- **CFMM with time-dependent curves (options-like trading)**
- **Central pool balance accounting and batch swapping functionality**
- **Non-trivial function approximations**
- **Use of assembly and inconsistent rounding methods**

=> Fuzz the swap function

Fuzz Test: Swapping Back And Forth

```
// Alice create's an honest pool with liquidity.
createPoolAndAllocateLiquidity(alice, 100, 10, 2e18, 1e18, 1_000_000 * 1e18);

// Bob create's a malicious pool with liquidity.
createPoolAndAllocateLiquidity(bob, volatility, duration, stkPrice, price, liquidity);

// Bob tries to extract tokens by swapping back and forth.
uint128 targetValue = 1e18;

// Swap input X -> tempOutput Y -> output X.
bool success = swapBackAndForth(sellAsset, input, tempOutput, input + targetValue);
if (!success) return;

// Bob withdraws all of his assets.
withdrawAllAssets(bob);

int256 assetGain = int256(asset.balanceOf(bob)) - int256(INITIAL_BALANCE);
int256 quoteGain = int256(quote.balanceOf(bob)) - int256(INITIAL_BALANCE);

// Bob should gain tokens.
if (assetGain < 0 || quoteGain < 0) return;

// Log the balance gain.
console.log("Asset gain", vm.toString(assetGain));
console.log("Quote gain", vm.toString(quoteGain));

// Report the test case.
fail();
```



[FAIL. Reason: UNDEFINED Counterex
[0] console::9710a9d0(
[staticcall]
[10850] HYPER::aa01380b(
[staticcall]
"9ed9f9cc:"
[14079] HYPER::
"EvmError:
"EvmError: Revert
FAILED. 0 passed,
over/underflow"
FAILED. 0 passed; 1 failed; 0 skipped; finished in 235.72ms

Refining the Testing Strategy

- **Address all reverts and bound parameters**

```
// Alice create's an honest pool with liquidity.
createPoolAndAllocateLiquidity(alice, 100, 10, 2e18, 1e18, 1_000_000 * 1e18);

// Bob create's a malicious pool with liquidity.
createPoolAndAllocateLiquidity(bob, volatility, duration, stkPrice, price, liquidity);

// Bob tries to extract tokens by swapping back and forth.
uint128 targetValue = 1e18;

// Swap input X -> tempOutput Y -> output X.
bool success = swapBackAndForth(sellAsset, input, tempOutput, input + targetValue);
if (!success) return;

// Bob withdraws all of his assets.
withdrawAllAssets(bob);

int256 assetGain = int256(asset.balanceOf(bob)) - int256(INITIAL_BALANCE);
int256 quoteGain = int256(quote.balanceOf(bob)) - int256(INITIAL_BALANCE);

// Bob should gain tokens.
if (assetGain < 0 || quoteGain < 0) return;

// Log the balance gain.
console.log("Asset gain", vm.toString(assetGain));
console.log("Quote gain", vm.toString(quoteGain));

// Report the test case.
fail();
```


Refining the Testing Strategy

- **Address all reverts and bound parameters**
- **Sanity check setup and improve coverage insight**

```
// Alice create's an honest pool with liquidity.
createPoolAndAllocateLiquidity(alice, 100, 10, 2e18, 1e18, 1_000_000 * 1e18);

// Bob create's a malicious pool with liquidity.
createPoolAndAllocateLiquidity(bob, volatility, duration, stkPrice, price, liquidity);

// Bob tries to extract tokens by swapping back and forth.
uint128 targetValue = 1e18;

// Swap input X -> tempOutput Y -> output X.
bool success = swapBackAndForth(sellAsset, input, tempOutput, input + targetValue);
if (!success) return;

// Bob withdraws all of his assets.
withdrawAllAssets(bob);

int256 assetGain = int256(asset.balanceOf(bob)) - int256(INITIAL_BALANCE);
int256 quoteGain = int256(quote.balanceOf(bob)) - int256(INITIAL_BALANCE);

// Bob should gain tokens.
if (assetGain < 0 || quoteGain < 0) return;

// Log the balance gain.
console.log("Asset gain", vm.toString(assetGain));
console.log("Quote gain", vm.toString(quoteGain));

// Report the test case.
fail();
```


Refining the Testing Strategy

- **Address all reverts and bound parameters**
- **Sanity check setup and improve coverage insight**
- **Question assumptions and conclusions in testing**

```
// Alice create's an honest pool with liquidity.
createPoolAndAllocateLiquidity(alice, 100, 10, 2e18, 1e18, 1_000_000 * 1e18);

// Bob create's a malicious pool with liquidity.
createPoolAndAllocateLiquidity(bob, volatility, duration, stkPrice, price, liquidity);

// Bob tries to extract tokens by swapping back and forth.
uint128 targetValue = 1e18;

// Swap input X -> tempOutput Y -> output X.
bool success = swapBackAndForth(sellAsset, input, tempOutput, input + targetValue);
if (!success) return;

// Bob withdraws all of his assets.
withdrawAllAssets(bob);

int256 assetGain = int256(asset.balanceOf(bob)) - int256(INITIAL_BALANCE);
int256 quoteGain = int256(quote.balanceOf(bob)) - int256(INITIAL_BALANCE);

// Bob should gain tokens.
if (assetGain < 0 || quoteGain < 0) return;

// Log the balance gain.
console.log("Asset gain", vm.toString(assetGain));
console.log("Quote gain", vm.toString(quoteGain));

// Report the test case.
fail();
```

Takeaway

- **Offensive testing requires a persistent, dynamic approach**
- **Aim to actively find potential cracks rather than just confirming robustness**
- **Question assumptions and validate your setup**

Stay in Touch

Full-length blog post

lovethewired.github.io/blog/2023/test-your-tests

phaze
@lovethewired

Questions?