Evasion Techniques

State of the Art

Web3

Apehex

04/08/2023

Known Techniques

1. Faking

1.1. Inheritance Overriding

1.1.1. Evades

Source code reviews with subtle exploitation of the compilation process.

1.1.2. How

The malicious contract inherits from standard code like *Ownable*, *Upgradeable*, etc.

It overwrites key elements by:

- adding a variable definition for an existing keyword
- polymorphism, which allows to have several versions of a function

Then a single keyword can refer to different implementations depending on its context.

The resulting contract doesn't behave like its parent, while looking legitimate.

1.1.3. Samples

Attribute Overwriting

KingOfTheHill inherits from *Ownable* but the original *owner* cannot be changed:

```
1  contract KingOfTheHill is Ownable {
2      address public owner; // different from the owner in Ownable
3
4      function () public payable {
5          if(msg.value > jackpot) owner = msg.sender; // local owner
6          jackpot += msg.value;
7      }
8      function takeAll () public onlyOwner { // owner from Ownable = contract
           creator
9          msg.sender.transfer(this.balance);
10         jackpot = 0;
11     }
12 }
```

In the modifier on *takeAll*, the *owner* points to the contract creator. It is at storage slot 1, while the fallback function overwrites the storage slot 2.

In short, sending funds to this contract will never make you the actual owner.

Method Overwriting

1.1.4. Detection & Countermeasures

- Caveat: these overrides appear in the sources but not in the bytecode.
- The sources can be checked for duplicate definitions / polymorphism.

Since the whole point is to advertize for a functionality with the sources, they will be available.

1.1.5. Resources

- [**paper-art-of-the-scam**], section *3.2.2*
- [**video-masquerading-code**]

1.2. Fake Standard Implementation

1.2.1. Evades

Etherscan's interpretation of proxy is fixed, it can easily be fooled.

1.2.2. How

Contrary to the previous methods, this one doesn't use valid code from the standards.
It keeps the name / structure, but the code is actually different.

1.2.3. Samples

Here's a fake EIP-1657 proxy implementation:

```
function _getImplementation() internal view returns (address) {
    return
        StorageSlot
            .getAddressSlot(bytes32(uint256(keccak256("eip1967.fake")) - 1)
                ).
            .value;
}
```

It doesn't use the standard slot for the implementation address: Etherscan will show some irrelevant contract, giving the impression it is legit.

1.2.4. Detection & Countermeasures

The bytecode selectors and implementation can be checked against reference implementations.

1.2.5. Resources

- [**video-masquerading-code**]

2. Morphing

2.1. Red-Pill

The red-pill technique detects simulation environment to disable its exploits upon scrutiny.

2.1.1. Evades

Live tests in transaction simulations: often performed by wallets before sending a transaction.

2.1.2. How

The contract detects simulation environments by:

- comparing the global variables with settings found in simulated environments:
  - *block.basefee* with
  - *block.coinbase* with *0x0000000000000000000000000000000000000000*
  - *tx.gasprice* with

Then it triggers legitimate code in simulation contexts and malicious code on the mainnet.

2.1.3. Samples

The contract [FakeWethGiveaway](red-pill/FakeWethGiveaway.sol) checks the current block miner's address:

```
1 function checkCoinbase() private view returns (bool result) {
2     assembly {
3         result := eq(coinbase(), 0x0000000000000000000000000000000000000000
              )
4     }
5 }
```

When null (test env), it actually sends a reward and otherwise it just accepts transfers without doing anything.s

2.1.4. Detection & Countermeasures

- Looking for unusual opcodes: typically 'block.coinbase'.
- Replaying transactions and fuzzing the global variables.

2.1.5. Resources

- [**article-red-pill**]

3. Obfuscation

3.1. Hiding In Plain Sight

3.1.1. Evades

Here, the goal is to overwhelm source code reviewers with the sheer volume of code.
It also lowers the efficiency of ML algorithms.

3.1.2. How

By stacking dependencies, the scammer grows the volume of the source code to thousands of lines.
99% of the code is classic, legitimate implementation of standards.
And the remaining percent is malicious code, hidden inside one of the numerous dependencies for example.

3.1.3. Samples

Hidden among 7k+ lines of code:

```
// no authorization modifier `onlyOwner`
function transferOwnership(address newOwner) public virtual {
    if (newOwner == address(0)) {
        revert OwnableInvalidOwner(address(0));
    }
    _transferOwnership(newOwner);
}
```

3.1.4. Detection & Countermeasures

1. The proportion of unused code can be leveraged from the transaction history.

3.2. Hiding Behind Proxies

3.2.1. Evades

- Etherscan code verification - source code reviews

3.2.2. How

Keeping the sources closed by only exposing a proxy contract.

3.3. Hidden State

3.3.1. Evades

Totally bypasses source & bytecode analysis by humans & tools.

3.3.2. How

At the construction / initialization, data can be put in storage at arbitrary slots.

3.3.3. Detection & Countermeasures

Detecting access to:

- arbitray storage locations
- locations given as input

4. Poisoning

4.1. Event Poisoning

5. Redirection

5.1. Hidden Proxy

5.1.1. Evades

This technique allows scammers to verify their contracts will dodging source code reviews.

5.1.2. How

The contract performs *delegateCalls* on any unknown selector.

The target address can be hardcoded, making it

In the end, the exposed functionalities are not meaningful, the logic is located at a seemingly unrelated address.

5.1.3. Samples

```
fallback () external {
  if (msg.sender == owner()) {
    (bool success, bytes memory data) = address(0
        x25B072502FB398eb4f428D60D01f18e8Ffa01448).delegateCall(
      msg.data
    );
  }
}
```

5.2. Selector Collisions

5.2.1. Evades

This subtle

5.2.2. How

Because the function selectors are only 4 bytes long, it is easy to find collisions.

When a selector in the proxy contract collides with another on the implementation side, the proxy takes precedence.

This can be used to override key elements of the implementation.

5.2.3. Samples

As shown in [the talk by Yoav Weiss at DSS 2023][video-masquerading-code]:

```
function IMGURL() public pure returns (bool) {
  return true;
}
```

This function has the same selector as *keccak("vaultManagers(address)")[0:4]*.

Foreseen Techniques

6. Obfuscation

6.1. Payload Packing

6.1.1. Evades

Pattern matching on the bytecode.

6.1.2. How

Encryption / encoding / compression can be leveraged to make malicious code unreadable.

6.1.3. Detection & Countermeasures

1. Scanning for high entropy data

Appendices

G. Samples

APPENDICES

H. Red Pill