MythX

REPORT 60FEE2BB0A209400195343FE

Created	Mon Jul 26 2021 16:28:43 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	60c9637f43f2c3c61312dd0c

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
af64738e-0071-4f36-9e16-4fc75d01b7e4	/contracts/wager.sol	6

Analysis af64738e-0071-4f36-9e16-4fc75d01b7e4 MythX

Started	Mon Jul 26 2021 16:28:53 GMT+0000 (Coordinated Universal Time)
Finished	Mon Jul 26 2021 16:31:02 GMT+0000 (Coordinated Universal Time)
Mode	Quick
Client Tool	Mythx-Vscode-Extension
Main Source File	/Contracts/Wager.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	6

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

/contracts/wager.sol

Locations

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.0;
3
4  import "./WagerFactory.sol";
```

LOW

SWC-107

A call to a user-supplied address is executed.

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

/contracts/wager.sol

Locations

```
161  internal
162  {
163  WagerFactory(factory).removeAddress(
164  uint(WagerFactory(factory).findIndexOfAddress(
165  address(this)
166  ))
167  );
168  }
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

/contracts/wager.sol

Locations

```
162 {
163 WagerFactory(factory).removeAddress(
164 uint(WagerFactory(factory).findIndexOfAddress(
165 address(this)
166 ))
167 );
168 }
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

/contracts/wager.sol

Locations

```
127 wagerAmount -= (wagerAmount * 7 / 1000);
128 randomNumber == 0 ?
129 payable(wagerer).transfer(wagerAmount) :
130 payable(challenger).transfer(wagerAmount);
```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

/contracts/wager.sol

Locations

```
128 randomNumber == 0 ?
129 payable(wagerer).transfer(wagerAmount) :
130 payable(challenger).transfer(wagerAmount);
131
132 wagerAmount = 0; // reset wager amount
```

LOW

SWC-116

A control flow decision is made based on The block.timestamp environment variable.

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/contracts/wager.sol

Locations

```
48
49  modifier validWagerAvailability {
50  require(block.timestamp < wagerExpireTime, "Wager has expired");
51  _;
52  }
```