Fishbank

Abstract

In this report, we consider the security of FishBank, who engaged ForsetiDev team on 23 January 2017 to perform smart contracts audit of FishBank. A audit was conducted on the commit version 5e645ad5c364413754f664b0a42e8d78a8059eb.

About ForsetiDev

ForsetiDev is a team of developers passionate about blockchain technology and specializing in smart contract development and security audits . Our core team members met at blockchain hackathon BlockchainHack2017 co-organized by Qtum. We created a Dispute Resolution Mechanism thats helps merchants make deals more transparent and won that hackathon. Since then we helped numerous projects conduct an ICO campaign,as well audited several smart contracts. Our team also won several hackathons : ICO lab Ecosystem hackathon и Latoken hackathon, as well had a honor to participate and to be among the winners of biggest ethereum hackathon so far EthWaterloo Hackathon.

Analysis technique

We used several publicly available automated Solidity analysis tools, as well as proceed manual analysis. All the issues found by tools were manually checked (rejected or confirmed). Contracts were manually analyzed, their logic was checked and compared with the one described in the whitepaper.

Bugs classification

CRITICAL - problems leading to stealing funds from any of the participants, or making them inaccessible by anyone

SEVERE - problems that can stop, freeze or break the internal logic of the contract

WARNING - non-critical problems that cannot break the contract, but contract code does not match declared in WhitePaper logic

NOTES - any other findings .

Automated Analysis

Securify

Transaction Reordering

Matched lines: L.382, L.345, L.413

Use of Untrusted Inputs in Security Operations

Matched lines: L.475, L.476, L.373, L.538, L.501, L.539, L.502, L.511

Block Number

Matched lines: L.475

All the issues found by tools were manually checked (rejected or confirmed). Cases, when these issues lead to actual bugs or vulnerabilities, are described in the next section.

Manual Analysis

Severe

Unauthorized access

FishBank.sol line 151

```
function getAquariumFee(bytes32 _hash) public {
 require(pendingFishing[_hash].deadline >= now);
```

if owner does not resolve hash the user can get the fee back after the deadline, but there is no check whether if msg.sender is the same that executed fishAquarium. The malicious user can scan for unresolved hashes and claim fee before owner

Send instead of transfer

Fishbank.sol, line 119
Fishbank.sol, line 156
Fishbank.sol, line 187

The send function is called inside checks instead of using transfer.
The recommended way to perform checked ether payments is addr.transfer(x), which automatically throws an exception if the transfer is unsuccessful.
Recomendation:
Use `transfer`, which is equivalent to `if (!send()) throw`;

In the following example, the send function is used:

```
if(!addr.send(1 ether)) {
revert();
}
```

Preferred alternative:

```
addr.transfer(1 ether);
```

Warnings

Transactions May Affect Ether Receiver

FishBank.sol, line 101

```
function setAquariumCost(uint256 _fee) onlyOwner public {
 aquariumCost = _fee;
}
```

A contract is exposed to this vulnerability if a miner (who executes and validates transactions) can reorder the transactions within a block in a way that affects the receiver of ether.
It's recommended to pause fishAquarium function before executing setAquariumCost

Extra value in bids is silently kept by the auction contract

ERC721Auction.sol, line 107

```
function buyAuction(uint256 _tokenId) payable external {
```

Any time a user bids on an auction with a larger than required amount, they will be shortchanged by the auction contract. Because the dutch auction model continuously reduces the price, and because there is an inevitable delay between sending a transaction and it being mined, there will almost always be excess funds.

Its recommended to modify buyAuction to return excess funds to the caller.

There is no return value for a function whose signature only denotes the type of the return value

Fishbank.sol, line 145

```
function fishAquariumAffiliate(uint256 _seed, address _affilitate, bool _fishBooster)
payable public returns (bytes32) {

  fishAquarium(_seed, _fishBooster);//create pending fishing

  pendingFishing[randomHashes[hashesUsed - 1]].affiliate = _affilitate;

}
```

Notes

Using the construction if (condition) {revert();} instead of require(condition);

Fishbank.sol, line 118
Fishbank.sol, line 156
Fishbank.sol, line 187
ERC721Auction.sol line 122
ERC721Auction.sol line 126
ERC721Auction.sol line 130

Since the construction *if (condition) {revert();}* is equivalent to *require(!condition);*
Recommendation: Use require for better code readability

Costly loop

Fishbank.sol, line 77
Fishbank.sol, line 124
Fishbank.sol, line 197
Fishbank.sol, line 215
Fishbank.sol, line 326

Loops are undesirable and quite dangerous in solidity, we recommend avoid them where it possible. If you can not avoid using cycles, you need to be very careful when using them.

Recommendations

Unchecked math

Fishbank.sol, line 252
Fishbank.sol, line 253
Fishbank.sol, line 283
Fishbank.sol, line 409

In the above places, we recommend to use “SafeMath”, to be sure.
Solidity is prone to integer over- and underflow. Overflow leads to unexpected effects and can lead to loss of funds if exploited by a malicious account.

Conclusion

In this report we have considered the security of smart contracts. The smart contracts have been analyzed under different aspects. There are some minor issues thats need to be considered,but overall, analysis showed a high quality of project, without critical vulnerabilities. The source code of the contracts is well documented and the methods are commented. No critical vulnerabilities were found during this audit