SOLIDIFIED

Audit Report for Fortunity. July 2nd, 2018.

Summary

Audit Report prepared by Solidified covering the Fortunity v2 contracts.

Process and Delivery

Two (2) independent Solidified experts performed an unbiased and isolated audit of the below contracts. The debrief took place on April 16, 2018 and the final results are presented here.

The current version, including follow up for previously reported issues was finalized in July 2nd, 2018.

Audited Contracts

The following contracts were covered during the audit:

- ICO
- BONUSROLL
- GAME

Notes

This audit was performed on a requested version of `FortunityFinalDraft` received via email.
This audit also considers comments and fixes made by Fortunity team after v1 audit.
This audit was based on the solidity compiler `0.4.21+commit.dfe3193c`

Issues Found

Wrong winner picked when the random number drawn is multiple of ticketPool

---

We assume that in the case of `ticketPool = 5`, for instance, the Wolfram Alpha query will be `random number between 1 and 50`. This was not completely clear from the spec, as the query itself is a parameter, and the deploy script was not provided. In the case where the random number generator yields the highest number possible, which is equal to `ticketPool * 10`, the raffle will erroneously pick the first buyer as the winner instead of the last. This is because the `windex` variable is set to `random % ticketPool`, and `50 % 5` is `0`, not `5`.

Additionally, if the owner create a contract with a random number query limit lower than ticketPool, the raffle is limited to the n first tickets sold (where n = the random query upper limit).

Recommendation

We recommend the raffle algorithm is reviewed in order to give all participants equal chance. If the random number query is to be passed as a parameter by the Owner, the raffle algorithm should not be affected by it in negative ways, like limiting the possible winners to a smaller group.

AMENDED [16 August 2018]:

Both issues above were fixed and are no longer present.

Contracts can be drained and locked in several ways by the Owner

---

The owner can perform a number of actions that compromise the trustlessness of the contracts. Although we understand the need for ownership functionality to perform operational tasks, in the current state the owner can perform functions that could be unreasonable and cause harm to users, as follows:

- All games added can transfer the whole balance of tokens owned by ICO contract. A regular user address can also be granted Game access.
- `takeAll` functions allow owner to drain ETH from contract, even during payout periods.
- `randomQuery` is passed as a parameter to the constructor of the lottery contract, and can be used to limit the group with chance to win).
- Owner can change the tokens' addresses at will (even during games).
- Fallback functions do not perform any actions (starting games, sending tokens back), they are payable and only call the `EthReceived` event.
- Owner can open and close payouts at will (there is a 7 day delay before being able to close the payouts) and users lose the amount if they do not call the contract during the payout.

Recommendation

Functions restricted to the owner should be kept to a minimum. All instances in which this access right could be used to harm users should be reviewed. it is acceptable for the owner to withdraw revenue from games, but the prize pool should remain untouched until after the payouts. Payout periods should be preset, and token contracts should remain locked during the lifecycle of a game. We also advise that the fallback functions are refactored to call the "buy" function of the respective contract.

PARTIALLY FIXED [16 August 2018]:

Fortunity acknowledges the issues outlined above, but prefers to maintain some of the trust points for operational reasons:

- Ether can still be drained at will by the owner;
- Ether from payouts is forfeit if not redeemed during the payout window. The owner still controls payout opening and closing within the 7 day period outlined above. Eth remaining from a payout round is intendedly spilled to the next round.

The following aspects of the issue above were fixed in the current version:

- Random query is no longer passed as a parameter and is now fixed (refer to issue #1).
- Functionality to change token contracts has been removed from the source code, they now have to be set on creation of games.
- Fallback functions now return Eth to users. Although a better solution, removing them would be beneficial to the contracts, since not even gas from the transaction would be charged from users.

Payout can be denied if the user didn't request for payout

Payouts are calculated and distributed when each user explicitly requests them using `requestDividendPayout`. If the user did not request for the payment during a payout duration, it cannot be claimed after the particular payout round. This is because of the admin's ability to change payout rounds without any restrictions.

Recommendation

It is a good practice to use withdraw pattern to distribute tokens. But it is recommended to keep track of users who did not request for a payout before next round.

AMENDED [12 June 2018]:

This issue has been partially fixed by the Fortunity team in the latest code shared. A cool-down period of 7 days is included which restricts the owner from closing the payout immediately. Comments from Fortunity team states that the payout denial is a design choice.

Input validations are not proper

The `transfer` and `transferFrom` functions in the ICO and BONUSROLL contracts are missing input validations. For example, verifying whether the target account is not `address(0)`.

Recommendation

Add input validations to all methods. It is strongly recommended to use ERC20 token contracts from the OpenZeppelin framework.

AMENDED [12 June 2018]:

This issue has been fixed by the Fortunity team and is not present in the latest code shared.

The game "steals" all overshooting ETH

If a game is initialized with `ticketPrice` = 0.1 ETH, and a user calls `buyTicket` with 0.3 ETH and `amount` set to 1, user only receives 1 ticket. The game takes the rest of user's ETH and does not give tickets for it.

Recommendation

The game should either refund the overshooting ETH to the buyer, or give the buyer the number of tickets according to how many ETH they send in (and refuse to receive an amount that does not give 0 for `msg.value % ticketprice`)

AMENDED [12 June 2018]:

This issue has been fixed by the Fortunity team and is not present in the latest code shared.

Consider the lack of support for floating point values

Since floating values are not fully supported by solidity, performing division before multiplication can lead to rounding errors.

Recommendation

Change the order of operations to perform multiplication before division.

AMENDED [12 June 2018]:

This issue has been fixed by the Fortunity team and is not present in the latest code shared.

Violation of Checks-Effects-Interactions pattern

Violation of a best practice pattern in `requestDividendPayout` and `startRoll`. The problem is not too serious here, because of the limited gas as part of `transfer`, but it is still recommended to use the pattern.

Recommendation

The pattern makes sure that you don't call an external function until you've done all the internal work you need to do. Implementation details can be found here.

AMENDED [12 June 2018]:

This issue has been fixed by the Fortunity team and is not present in the latest code shared.

Validation is performed before price calculation

Price is calculated after checking for user balance. It is recommended to declare a variable before its usage to improve the readability. Future versions of solidity may use C99 scoping rules to strictly check similar declarations.

AMENDED [12 June 2018]:

This issue has been fixed by the Fortunity team and is not present in the latest code shared.

Use OpenZeppelin contracts

It is recommended to use well-tested open source libraries for achieving common functionalities. OpenZeppelin is a popular library that provides a set of reusable contracts. It is suggested to use the following contracts from the OpenZeppelin library.

- Ownership Contract
- ERC20 Contracts

Use require() over if

Consider using `require` over `if-revert` for validations if the logic is not complex.

AMENDED [12 June 2018]:

This issue has been fixed by the Fortunity team and is not present in the latest code shared.

SOLIDIFIED

Audit Report for Fortunity. July 2nd, 2018.

Closing Summary

Several issues were found during the second audit of Fortunity contracts that could break the intended behavior. It is highly recommended that the Fortunity team addresses all Critical, Major and Minor issues reported and performs a follow up verification of fixes with an auditing group, followed by a bug bounty program.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of the Fortunity platform or its products. This audit does not provide a security or correctness guarantee of the audited smart contracts. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.